针对初中级开发人员，讲透HTML与CSS核心知识，兼顾高级技巧与开发思想，助你达到真正前端工程师的实战开发水平。

Web前端开发精品课
HTML与CSS进阶教程

莫振杰 编著

含金量高 前端精品内容荟萃，强化基础提升实战技能。

通俗易懂 语言风格轻松幽默，讲解枯燥知识形象生动。

系统学习 掌握前端高级技巧，学习进阶内容清晰流畅。

贴近读者 结合自身学习经历，文字极具温度不失严谨。

直击痛点 规避开发思维误区，精炼浓缩直指技术本质。

中国工信出版集团 人民邮电出版社 POSTS & TELECOM PRESS

目　录

版权信息

书名：Web前端开发精品课 HTML与CSS进阶教程

ISBN：978-7-115-43295-7

- 著　　　　莫振杰

  责任编辑　赵 轩

- 人民邮电出版社出版发行　　北京市丰台区成寿寺路11号

  邮编　100164　电子邮件　315@ptpress.com.cn

  网址　http://www.ptpress.com.cn

- 读者服务热线：(010)81055410

  反盗版热线：(010)81055315

内容提要

本书内容结合笔者在前后端大量开发中的实战经验，系统化知识，浓缩精华，用通俗易懂的语言直击学习者的痛点。通过本书，能让你从“野生网页设计师”水平提升达到“真正前端工程师”水平。

全书分为两大部分：第一部分是HTML进阶内容，主要介绍HTML高级技巧和HTML语义化；第二部分是CSS进阶内容，主要介绍CSS开发技巧、代码规范、性能优化、属性本质、重要概念（如包含块、BFC和IFC等）。

除了知识讲解，本书还融入了大量的开发案例，更加注重编程思维的培养，并且提供学习者一个流畅的学习思路。

前言

近年来，Web前端技术飞速发展，作用也日趋重要。在Web前端技术中，HTML和CSS是最基础的知识。当前市面上同类书籍很多，但大部分都是介绍基础性的HTML标签和CSS属性，缺乏一本真正体现前端开发水平的书籍。

本书是《Web前端开发精品课HTML和CSS基础教程》的姊妹篇，对于基础知识，不再进行详细介绍。HTML和CSS知识多而杂，因此入门容易精通难。本书对高级部分的内容进行深加工，使其更加完善。为了避免大家多走弯路，笔者把前端碎片化的知识系统化，提供给学习者一个流畅的学习思路。

本书特点

- 系统全面，进阶内容清晰梳理，便于读者流畅地学习。
- 通俗易懂，用最贴心形象的语言直指技术本质。
- 贴近读者，结合自己学习经历，文字有温度。
- 直击痛点，规避学习中的很多思维误区。
- 含金量高，前端高质量内容整合，呈现高级开发技巧。

读者对象

- 有一定基础的Web前端工程师
- 大中专院校相关专业学生

特别致谢

在编写本书的过程中，笔者得到了很多人的帮助。首先感谢陈志东先生花费大量时间对本书进行细致的审阅，给出诸多非常棒的建议，并且为本书绘制了大量精美的插图。

感谢充满创意和活力的五叶草团队（陈志东、邵婵、程紫梦）一直以来的陪伴和支持。我的人生因为有你们而更加精彩。

感谢赵轩老师（本书责任编辑）在这段时间给予我细心的指导和不懈的支持，有您的帮助本书才得以顺利出版。

由于笔者水平有限，书中难免有错漏之处，读者如果遇到问题或有意见和建议，可以到绿叶学习网（www.lvyestudy.com）或者发邮件（lvyestudy@foxmail.com）与我联系。

第一部分　HTML进阶

第1章　HTML基础知识

1.1　HTML和CSS进阶简介

1.1.1　你真的精通HTML和CSS吗

我们都知道，前端技术最核心的三大技术是HTML、CSS和JavaScript。HTML定义网页的结构，CSS定义网页的外观，而JavaScript定义页面的行为。其中HTML和CSS是前端技术中最基础的东西。

HTML和CSS入门容易，精通却很难，特别是CSS。“什么？精通很难？我很确定我已经精通CSS了啊！”这种话往往是出自学习三两个月、技术刚入门的人之口。我就曾经碰到不少这样的人。对于HTML来说，确实没有多少东西可以深入的，但是CSS却不一样。

如果你认为自己精通HTML和CSS了，那你可以思考一下下面这些问题。

（1）什么是HTML语义化？对于标题、图片、表格、表单，如何更好地实现这些方面的语义化？

（2）什么是外边距叠加？出现外边距叠加的根本原因是什么？

（3）在CSS中，关于命名、书写以及注释都有哪些好的规范（便于团队开发和后期维护）？

（4）说一下display这几个属性值的区别：block、inline、inline-block、table-cell。

（5）你深入了解过text-indent、text-align、line-height以及vertical-align这几个属性么？它们都有哪些高级技巧。

（6）为什么overflow:hidden可以清除浮动？

（7）CSS都有哪些性能优化技巧？如何使用更高性能的选择器？

（8）如何使用CSS实现各种图形效果，例如三角形、圆、椭圆等？

（9）解释一下这几个概念：包含块、BFC和IFC、层叠上下文。

……

如果你有一半答不上来，说明你连“熟悉CSS”都算不上，更别说“精通CSS”了。因此大家不要学了几个标签就认为自己精通HTML，也不要学了几个属性就觉得自己精通CSS了。不管是哪门技术，自己都应该深入地去学习，自我满足只会让自己滞留不前。

1.1.2　进阶教程简介

HTML进阶的内容只针对HTML 4.01，而CSS进阶的内容只针对CSS 2.1。对于HTML 5和CSS 3的内容，可以关注绿叶学习网。

本书是《Web前端开发精品课HTML和CSS基础教程》的姊妹篇，两者具有很强的连贯性。本书并不适合没有基础的人学习，对于HTML和CSS入门的相关知识，可以参考本书的姊妹篇，不然在学习本书的过程中可能对有些东西无法理解。

本书虽然涉及的东西很多，但浓缩的都是精华。有一句话说得好：“干扰因素越少，越容易专注一件事”，因此对于书中的技巧我们也会以最简单的例子来讲解。笔者在编写本书的时候也是字斟句酌，该展开的会详细展开，没用的东西一定会一笔带过。希望大家在学习中不要跳跃性地学习。

此外，本书里很多东西比较复杂，一时半会儿可能消化不了，应该多回来翻几遍，并且结合自己的实践来理解。“书读百遍，其义自见。”古人有些话还是说得蛮好的。HTML和CSS这些进阶知识在本书中已经梳理得比较完善了，学习中可以来回翻一番，想当年我们连“翻”的份都没有，因为压根儿就没有这样的一个系统化的学习教程。

对于本书的学习，一定要下载这本书的源代码，一边查看源代码，一边测试效果。本书中的代码请从异步社区www.epubit.com.cn本书页面下载。

1.2　HTML、XHTML和HTML5

很多新手往往分不清HTML、XHTML和HTML5，这一节给大家详细讲解一下这三者的关系和区别。

1.2.1　HTML和XHTML

HTML，全称HyperText Mark-up Language（超文本标记语言），是构成网页文档的主要语言。我们常说的HTML指的是HTML 4.01。

XHTML，全称EXtensible HyperText Mark-up Language（扩展的超文本标记语言），它是XML风格的HTML 4.01，我们可以称之为更严格、更纯净的HTML 4.01。

HTML语法书写比较松散，利于开发者编写。但是对于机器，如电脑、手机等来说，语法越松散，处理起来越困难。因此为了让机器更好地处理HTML，我们才在HTML基础上引入了XHTML。

XHTML相对于HTML来说，在语法上更加严格。XHTML和HTML主要区别如下。

1．XHTML标签必须闭合。

在XHTML中，所有标签必须闭合，例如“<p></p>”“<div></div>”等。此外，空标签也需要闭合，例如
要写成
。

错误写法：<p>欢迎来到绿叶学习网

正确写法：<p>欢迎来到绿叶学习网</p>

2．XHTML标签以及属性必须小写。

在XHTML中，所有标签以及标签属性必须小写，不能大小写混合，也不能全部都是大写。不过标签的属性值可以大写。

错误写法：<Body><DIV></DIV></Body>

正确写法：<body><div></div></body>

3．XHTML标签属性必须用引号。

在XHTML中，标签属性值必须用引号括起来，单引号、双引号都可以。

错误写法：<input id=txt type=text/>

正确写法：<input id="txt" type="text"/>

4．XHTML标签用id属性代替name属性。

在XHTML中，除了表单元素之外的所有元素，都应该用id而不是name。

错误写法：<div name="wrapper"></div>

正确写法：<div id="wrapper"></div>

下面是一个完整的XHTML文档。

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>web前端开发精品课系列</title>
</head>
<body>
    <p>《web前端开发精品课·HTML和CSS基础教程》</p>
    <p>《web前端开发精品课·HTML和CSS进阶教程》</p>
</body>
</html>
```

1.2.2　HTML5

HTML指的是HTML 4.01，XHTML是HTML的过渡版本，XHTML是XML风格的HTML 4.01。而HTML 5指的是下一代的HTML，也就是

HTML 4.01的升级版。

不过HTML 5已经不再是单纯意义上的标签了，它已经远远超越了标签的范畴。HTML 5除了新增部分标签之外，还增加了一组技术，包括canvas、SVG、WebSocket、本地存储等。这些新增的技术都是使用JavaScript来操作。也就是说，HTML 5使得HTML从一门“标记语言”转变为一门“编程语言”。

对于HTML 5中的新技术，在此不做详细介绍。单纯从新增的标签上来看，HTML 5有以下几个特点。

1. 文档类型说明

基于HTML 5设计准则中的“化繁为简”原则，页面的文档类型<!DOCTYPE>被极大地简化了。

XHTML文档声明如下：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.
```

HTML5文档声明如下：

```
<!DOCTYPE html>
```

2. 标签不再区分大小写

```
<div>绿叶学习网</DIV>
```

上面这种写法也是完全符合HTML5规范的。但是在实际开发中，建议所有标签以及属性都采用小写方式。

3. 允许属性值不加引号

```
<div id=wrapper style=color:red>绿叶学习网</div>
```

上面这种写法也是完全符合HTML5规范的。但是在实际开发中，

建议标签所有属性值都加引号，单引号或双引号都可以。

4．允许部分属性的属性值省略

在HTML5中，部分具有特殊性属性的属性值是可以省略的。例如，下面代码是完全符合HTML 5规范的：

```
<input type="text" readonly/>
<input type="checkbox" checked/>
```

上面两句代码等价于：

```
<input type="text" readonly="readonly"/>
<input type="checkbox" checked="checked"/>
```

在HTML 5中，可以省略属性值的属性如表1-1所示。

表**1-1 HTML5**中可以省略属性值的属性

省 略 形 式	等　价　于
checked	checked="checked"
readonly	readonly="readonly"
defer	defer="defer"
ismap	ismap="ismap"
nohref	nohref="nohref"
noshade	noshade="noshade"
nowrap	nowrap="nowrap"

selected	selected="selected"
disabled	disabled="disabled"
multiple	multiple="multiple"
noresize	noresize="disabled"

一句话概括HTML、XHTML和HTML 5就是：HTML指的是HTML 4.01，XHTML是HTML的过渡版，HTML 5是HTML的升级版。

1.3　div和span

对于div和span这两个元素，不少新手也不知道它们之间有什么区别，使用起来也很随便。因此，这里有必要简单介绍一下。

div和span没有任何语义，正是因为没有语义，这两个标签一般都是配合CSS来定义元素样式的。

div和span区别如下。

（1）div是块元素，可以包含任何块元素和行内元素，不会与其他元素位于同一行；span是行内元素，可以与其他行内元素位于同一行。

（2）div常用于页面中较大块的结构划分，然后配合CSS来操作；span一般用来包含文字等，它没有结构的意义，纯粹是应用样式。当其他行内元素都不适合的时候，可以用span来配合CSS操作。

其实，除了div和span外，还有一个label标签。div和span是无语义标签，但label是有语义标签。label只适用于表单中，用于显示在输入控件旁边的说明性文字。关于label标签，我们在后面“2.5　表单语义化”一节会详细介绍。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
</head>
<body>
    <p>"<span style="font-weight:bold;color:Red;">视觉化思考</span>"能以独特
</body>
</html>
```

在浏览器预览效果如图1-1所示。



图1-1　div和span

分析：

在这个例子中，我们想要对“视觉化思考”这几个文字加粗或者改变颜色，此时可以使用span包含文字，然后再进行样式修改。事实上，span标签往往都是用来配合CSS来修饰元素的。

对于div和span，大家经过一些实践，自然而然会有更深刻的理解，在此就不啰嗦了。

1.4　id和class

id和class是HTML元素中两个最基本的公共属性。一般情况下，id和class都用来选择元素，以便进行CSS操作或者JavaScript操作。但是很多新手对id和class这两个属性感到很迷茫，不知道什么时候用id，什么时候用class，甚至随便使用。

1.4.1　id属性

id属性具有唯一性，也就是说在一个页面中相同的id只允许出现一次。W3C建议，对于页面关键的结构或者大结构，我们才使用id。所谓的关键结构，指的是诸如LOGO、导航、主体内容、底部信息栏等结构。对于一些小地方，还是建议使用class属性。

我们知道搜索引擎识别一个页面结构，是根据标签的语义以及id属性来识别的。因此id属性不要轻易使用。此外，id的命名也十分关键，特别是对搜索引擎优化而言。对于id和class的命名，我们在CSS进阶部分会详细介绍。

1.4.2　class属性

class，顾名思义，就是“类”。它采用的思想跟C、Java等编程语言中的“类”相似。我们可以为同一个页面的相同元素或者不同元素设置相同的class，然后使得相同class的元素具有相同的CSS样式。

如果你要为两个或者两个以上元素定义相同的样式，建议使用class属性。因为这样可以减少大量的重复代码。

注意，对于一个元素而言，我们可以定义多个class。一般来说，定义多个class的目的在于：一般用一个class抽取公共样式，然后用另外一个class定义单独样式。

举个例子，在一个页面中有如图1-2所示的三个栏目，仔细分析我们会发现这三个栏目具有部分相同样式，此时我们可以使用一个class来定义公共样式，但是这三个栏目又有各自的样式。该怎么办呢？这时候

我们应该为这三个栏目分别定义不同的class，以便在CSS中控制单独的样式。这就是多个class的用处。



图1-2　绿叶学习网中三个栏目

上图的三个栏目的HTML关键结构如下：

```
<div class="column blog">
     <h2></h2>
     <div></div>
</div>
<div class="column manual">
     <h2></h2>
     <div></div>
</div>
<div class="column tool">
     <h2></h2>
     <div></div>
</div>
```

对于id和class，我们总结一下：对于页面关键结构，建议使用id；对于小地方，建议使用class。就算我们不需要对关键结构进行CSS操作或者JavaScript操作，也建议加上id，以便搜索引擎识别页面结构。

1.5 浏览器标题栏小图标

在浏览网页的时候，我们会发现几乎所有网站的页面在浏览器标题栏前面都会有一个小图标，如图1-3所示。

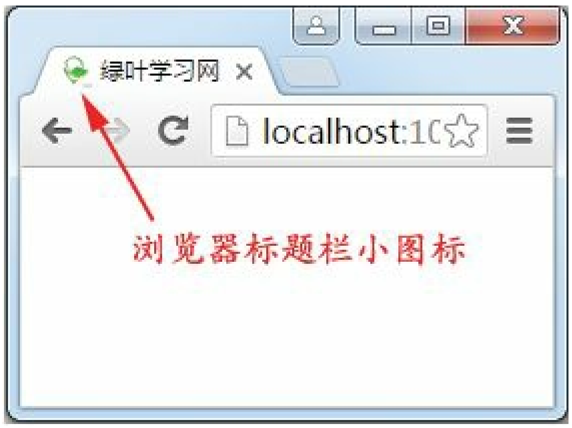


图1-3　浏览器标题栏小图标

想要实现这个效果，我们只需要在head标签添加一个link标签即可。

语法：

```
<link rel="shortcut icon" type="image/x-icon" href="favicon.ico"/>
```

说明：

rel和type这两个属性的取值是固定形式，无需多讲。href属性取值为小图标的地址，这个地址是根据小图标在站点文件夹路径而定的，跟图片引用路径是一样的道理。

这里注意一下，小图标格式是.ico，而不是.jpeg、.png、.gif等格式。对于.ico格式的图标制作，我们可以搜索一下“在线icon”，会发现很多不错的在线工具，大家可以收藏一下。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>绿叶学习网</title>
    <link rel="shortcut icon" type="image/x-icon" href="images/favicon.ico"
</head>
<body>
</body>
</html>
```

在浏览器预览效果如图1-4所示。

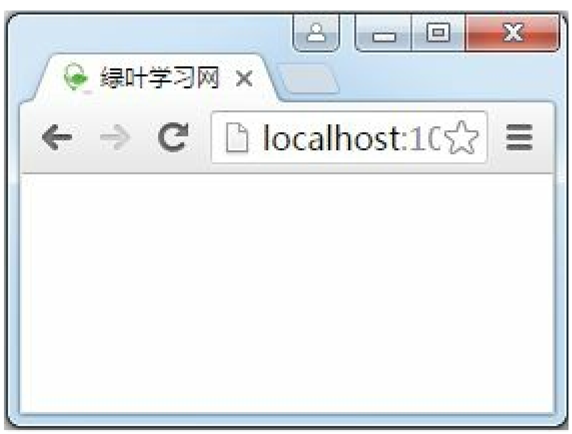


图1-4　浏览器标题栏小图标预览图

第2章　语义化

2.1　语义化简介

由于HTML简单，很多初学者对它存在一种偏见，觉得它没多少东西，因此在学习的过程中随便对待。其实，学习HTML的重点不在于掌握了多少标签，而是在于掌握标签的语义以及如何编写一个语义结构良好的页面。

在实际开发过程中，很多人由于对标签语义不熟悉，常常用某一个标签代替另外一个标签来实现某些效果。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title></title>
     <style type="text/css">
           body{font-family:"微软雅黑";font-size:14px;}
           .content
           {
                width:300px;
                padding:10px;
                border:1px dashed gray;
           }
           .content div
           {
                font-size:16px;
                font-weight:bold;
                height:24px;
```

```
                line-height:24px;
            }
    </style>
</head>
<body>
    <div class="content">
        <div>web前端开发</div>
        <p>web前端开发最核心3个技术：HTML、CSS和JavaScript。HTML控制网页的结构，
```

</div>

</body>

</html>

在浏览器预览效果如图2-1所示。



图2-1　div实现的标题效果

对于上面的标题效果，正确的做法应该是使用h1～h6标签来实现，但这里却使用div来代替了。虽然页面效果一样，但是这种“用某一个标签代替另外一个标签来实现相同效果”的做法是完全不可取的，因为它违背了HTML这门语言的初衷。

HTML的精髓就在于标签的语义。在HTML中，大部分标签都有它自身的语义，例如p标签，表示的是“paragraph”，标记的是一个段落；h1标签，表示“header1”，标记的是一个最高级标题……而div和span是无语义的标签，我们应该尽可能少用。

HTML很简单，因此很多初学者往往忽略了它的目的和重要性。我们学习HTML并不是看自己学了多少标签，更重要的是在你需要的地方能否用到正确的语义化标签。把标签用在对的地方，这才是HTML学习的目的所在。

我们都知道前端最核心的技术是HTML、CSS和JavaScript这三种。其中HTML是网页的结构，CSS是网页的外观，JavaScript是网页的行为。在这三大元素中，HTML才是最重要的，而CSS和JavaScript只是用来修饰结构的。就像你盖房子，房子装饰得再漂亮，如果结构不稳也容易塌。

整站开发时，编写的代码往往都是成千上万行，如果我们全部使用div和span来代替语义化标签，后期维护会非常困难。此外对于一个页面来说，我们可以根据一个页面的外观来判断哪些是标题，哪些是图片。但是搜索引擎跟人不一样，它可“看不懂”一个页面长什么样的。它只会根据HTML代码来识别。搜索引擎一般都是根据HTML标签来识别这里是一个img标签，那里是一个p标签等。如果整个页面都是div和span，搜索引擎小蜘蛛肯定会迷路，可能以后都不想来光顾你这个站点。要是这样的话，你崩溃了，你的老板也跟着崩溃了。

从上面我们知道，编写一个语义结构良好的页面在实际开发中极其重要。主要有两个最大的优点：①利于开发调试和后期维护；②利于搜索引擎优化。在这一章，我们从以下7个方面给大家详细讲解HTML语义化。

（1）标题语义化。

（2）图片语义化。

（3）表格语义化。

（4）表单语义化。

（5）其他语义化。

（6）语义化验证。

（7）HTML 5摒弃的标签。

2.2　标题语义化

h1～h6是标题标签，h表示“header”。h1～h6在HTML语义化中占有极其重要的地位。h1～h6按照标题的重要性依次递减，其中h1重要性最高，h6重要性最低。

相对于其他语义化标签，h1～h6在搜索引擎优化（即SEO）中占有相当重要的地位。在一个页面中，h1～h6这6个标签，我们不需要全部都用上，都是根据需要才用的。对于h1～h6，一般情况下我们只会用到h1、h2、h3和h4，很少再会去用h5和h6，因为一个页面不可能用到那么多级的标题。并且从搜索引擎优化的角度来说，h1、h2、h3和h4这4个标签会被赋予一定的权重，而h5和h6的权重跟普通标签差不多，在SEO优化上意义不大。

对于标题h1～h6的语义化，我们需要注意以下四个方面。

（1）一个页面只能有一个h1标签。

（2）h1～h6之间不要断层。

（3）不要用h1～h6来定义样式。

（4）不要用div来代替h1～h6。

1．一个页面只能有一个h1标签

h1标签表示每个页面中最高层级的标题，搜索引擎会赋予h1标签最高权重。虽然W3C没有明确规定一个页面不能有多个h1标签，但是我们还是推荐“一个页面一个h1标签”的做法。如果一个页面出现多个h1，对搜索引擎可能不好，也可能会被判以作弊。就像你写文章一样，一个页面就等于一篇文章，你见过一篇文章有多个主标题的么？

2．h1～h6之间不要出现断层

搜索引擎对h1～h6标签比较敏感，尤其是h1和h2。一个语义良好的

页面，h1～h6应该是完整有序而没有出现断层的。也就是说，要按照“h1、h2、h3、h4”这样的顺序依次排列下来，不要出现“h1、h3、h4”而漏掉h2的情况。

3．不要用h1～h6来定义样式

我们都知道h1～h6是有默认样式的，如图2-1所示。在实际开发中，很多时候我们需要为文本定义字体加粗或者字体大小。有些人喜欢用h1～h6来代替CSS，使用标签来控制样式，这是一种非常不好的做法。我们一定要记住，HTML关注的是结构（语义），CSS关注的是样式，结构跟样式应该分离。



图2-2　h1～h6在浏览器中的效果

4．不要用div来代替h1～h6

从语义上来说，一个页面的标题应该使用h1～h6标签，不要使用div来代替。

举例：

分析：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title></title>
    <style type="text/css">
        body{font-family:"微软雅黑";font-size:14px;}
        .content
        {
            width:300px;
            padding:10px;
             border:1px dashed gray;
        }
        .content div
        {
            font-size:16px;
            font-weight:bold;
            height:24px;
            line-height:24px;
        }
    </style>
</head>
<body>
    <div class="content">
        <div>web前端开发</div>
        <p>web前端开发最核心3个技术：HTML、CSS和JavaScript。HTML控制网页的结构，
    </div>
</body>
</html>
```

在浏览器预览效果如图2-3所示。

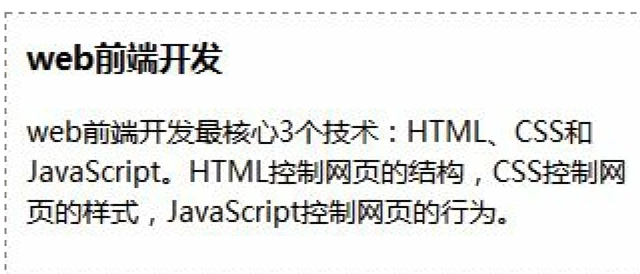


图2-3　div实现的标题效果

分析：

div是无语义的标签，如果使用div来代替h1～h6，后期维护比较困难，而且对SEO影响也非常大。因为这种做法会让一个页面丢失大量的权重。

2.3　图片语义化

在HTML中，我们使用img标签来表示图片。对于图片的语义化，我们从以下两个方面来深入探讨一下。

（1）alt属性和title属性。

（2）figure元素和figcaption元素。

2.3.1　alt属性和title属性

img标签有两个重要属性：alt和title。

alt属性用于图片描述，这个描述文字是给搜索引擎看的。并且当图片无法显示时，页面会显示alt中的文字。

title属性也用于图片描述，不过这个描述文字是给用户看的。并且当鼠标指针移到图片上时，会显示title中的内容。

语法：

```
<img src="" alt="图片描述(给搜索引擎看)" title="图片描述(给用户看)">
```

说明：

搜索引擎跟人不一样，它看不出一张图片描绘的是什么东西，它只会查看HTML代码，通过img标签的alt属性或者页面上下文来判断图片的内容。因此，对于img标签，我们一定要添加alt属性，以便搜索引擎识别图片的内容。alt属性在搜索引擎优化中也很重要，并且会被赋予一定的权重。

请一定要注意：alt属性是img标签必需属性，一定要添加；title属性是img标签可选属性，可加可不加。建议大家在实际开发中，对于img标签，要记得在alt属性中添加必要的描述信息。

2.3.2 figure元素和figcaption元素



图2-4 “图片+图注”效果

对于如图2-4所示的这种“图片+图注”的效果，我们可以使用如下代码来实现。

```
<div class="img-list">
     <img src="" alt=""/>
     <span>HTML入门教程</span>
<div>
```

但是这种实现方式的语义并不好。在HTML 5中，引入了figure和figcaption两个元素来增强图片的语义化。

语法：

```
<figure>
    <img src="" alt=""/>
    <figcaption></figcaption>
</figure>
```

说明：

figure元素用于包含图片和图注，figcaption元素用于表示图注文字。在实际开发中，对于“图片+图注”效果，我们都建议使用figure和figcaption这两个元素来实现，从而使得页面的语义更加良好。

2.4　表格语义化

不少初学者看了《Web前端开发精品课HTML和CSS基础教程》（本书的姊妹篇）跑来问：“不是说表格布局已经被抛弃了吗？为什么还要在书里讲解表格呢？这不是多此一举吗？”其实不然，在实际开发中，我们不建议使用表格布局，应该使用浮动布局或者定位布局。虽然表格拿来做布局的方式被抛弃了，但是这并没有说明表格就一无是处了。

问大家一个问题：如图2-5所示的这种表格数据的展示，应该怎么实现呢？不少得了“table恐惧症”的小伙伴可能会想到使用div来模拟表格。事实上，对于这种表格数据形式，最好的选择还是table。



font-weight属性

font-weight属性值	说明
normal	默认值，正常体
lighter	较细
bold	较粗
bolder	很粗（其实效果跟bold差不多）

图2-5　绿叶学习网中的表格

在表格中，我们比较常用的标签是table、tr和td这3个。不过为了加强表格的语义化，W3C还增加了5个标签：th、caption、thead、tbody和tfoot。th表示“表头单元格”；caption表示“表格标题”。thead、tbody和tfoot这3个标签把表格从语义上分为三部分：表头、表身和表脚。有了这几个标签，表格语义更加良好，结构更加清晰。对于这5个标签，我们在本书的姊妹篇中已经做了详细的介绍，这里就不再赘述。表格标签如表2-1所示。

表**2-1** 表格标签

标　　签	说　　明

table	表格
caption	标题
thead	表头（语义划分）
tbody	表身（语义划分）
tfoot	表尾（语义划分）
tr	行
th	表头单元格
td	表格单元格

语法：

```
<table>
    <caption>表格标题</caption>
    <!--表头-->
    <thead>
        <tr>
            <th>表头单元格1</th>
            <th>表头单元格2</th>
        </tr>
    </thead>
    <!--表身-->
    <tbody>
        <tr>
            <td>标准单元格1</td>
            <td>标准单元格2</td>
        </tr>
        <tr>
```

```
            <td>标准单元格1</td>
            <td>标准单元格2</td>
        </tr>
    </tbody>
    <!--表脚-->
    <tfoot>
        <tr>
            <td>标准单元格1</td>
            <td>标准单元格2</td>
        </tr>
    </tfoot>
</table>
```

说明：

thead、tbody和tfoot这三个标签也是表格中非常重要的标签，它从语义上区分了表头、表身和表脚。很多人容易忽略这三个标签。

上述语法显示效果如图2-6所示。

表格标题

表头单元格1	**表头单元格2**
标准单元格1	标准单元格2
标准单元格1	标准单元格2
标准单元格1	标准单元格2

图2-6　表格标签

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        table, thead, tbody, tfoot, th, td
        {
            border: 1px dashed gray;
        }
    </style>
</head>
<body>
```

```
    <table>
        <caption>考试成绩表</caption>
        <thead>
            <tr>
                <th>姓名</th>
                <th>语文</th>
                <th>英语</th>
                <th>数学</th>
            </tr>
        </thead>
        <tbody>
            <tr>
                <td>小明</td>
                <td>80</td>
                <td>80</td>
                <td>80</td>
            </tr>
            <tr>
                <td>小红</td>
                <td>90</td>
                <td>90</td>
                <td>90</td>
            </tr>
            <tr>
                <td>小杰</td>
                <td>100</td>
                <td>100</td>
                <td>100</td>
            </tr>
        </tbody>
        <tfoot>
            <tr>
                <td>平均</td>
                <td>90</td>
                <td>90</td>
                <td>90</td>
            </tr>
        </tfoot>
    </table>
</body>
</html>
```

在浏览器预览效果如图2-7所示。

考试成绩表

姓名	语文	英语	数学
小明	80	80	80
小红	90	90	90
小杰	100	100	100
平均	90	90	90

图2-7　表格语义化实例

分析：

对于thead、tbody和tfoot这3个标签，不一定能够全部都用得上，例如tfoot就很少用。一般情况下，我们都是根据实际需要来使用这3个标签的。

2.5　表单语义化

表单跟表格，这是两个完全不一样的概念，不过还是有不少初学者傻傻分不清。对于表单语义化，我们从2个方面来探究一下。

（1）label标签。

（2）fieldset标签和legend标签。

对于图2-8所示效果，我们可能会使用如下HTML代码来实现。

```
<form action="index.aspx" method="post">
    <div>登录绿叶学习网</div>
    <p>
        <span>账号：</span><input type="text" name="name" />
    </p>
    <p>
        <span>密码：</span><input type="password" name="pwd" />
    </p>
    <input type="checkbox" name="remember-me" /><span>记住我</span>
    <input type="submit" value="登录" />
</form>
```

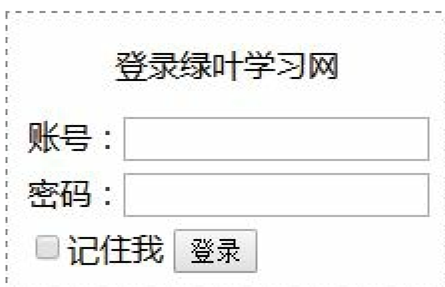


图2-8　表单效果

2.5.1　label标签

W3C规范定义，label标签用于显示在输入控件旁边的说明性文字。也就是将某个表单元素和某段说明文字关联起来。

语法：

```
<label for="">说明性文字</label>
```

说明：

label标签for属性值为所关联的表单元素的id，例如<input id="name" type="text"/>，则其所关联的label标签应该为<label for="name"></lable>。

label标签的for属性有两个作用。

（1）语义上绑定了label元素和表单元素。

（2）增强了鼠标可用性。也就是说我们点击label中的文本时，其所关联的表单元素也会获得焦点。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
</head>
<body>
    <div>
        <input id="Radio1" type="radio"/>单选框
        <input id="Checkbox1" type="checkbox" />复选框
    </div>
    <hr/>
    <div>
        <input id="rdo" name="rdo" type="radio"/><label for="rdo">单选框</la
        <input id="cbk" name="cbk" type="checkbox" /><label for="cbk">复选框
    </div>
```

```
</body>
</html>
```

在浏览器预览效果如图2-9所示。

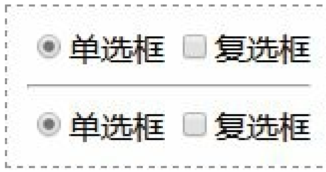


图2-9　label标签与单选框/复选框

分析：

从这个例子可以看到，在第一组表单中，我们只能点击单选框才能选中单选框，点击它旁边的说明文字是不能选中的。在第二组表单中，我们可以点击单选框来选中单选框，并且点击它旁边的说明文字同样也可以选中单选框。而对于复选框来说，也是一样的效果。

其实，这就是label标签for属性的作用。for属性使得鼠标单击的范围扩大到label元素上，极大地提高了用户单击的可操作性。事实上，label标签有两种关联方式，我们拿复选框来说，下面两行代码是等价的。

```
<input id="cbk" type="checkbox" /><label for="cbk">复选框</label>
<label>复选框<input id="cbk" type="checkbox"/></label>
```

对于图2-8中的效果，我们使用label标签来增强语义化，修改后的代码如下。

```
<form action="index.aspx" method="post">
    <div>登录绿叶学习网</div>
    <p>
        <label for="name">账号：</label><input type="text" id="name" name="n
    </p>
    <p>
        <label for="pwd">密码：</label><input type="password" id="pwd" name=
    </p>
```

```
    <input type="checkbox" id="remember-me" name="remember-me"/><label for=
    <input type="submit" value="登录"/>
</form>
```

2.5.2 fieldset标签和legend标签

在表单中，我们还可以使用fieldset标签来给表单元素进行分组。其中，legend标签用于定义某一组表单的标题。

语法：

```
<fieldset>
    <legend>表单组标题</legend>
    ……
</fieldset>
```

说明：

使用fieldset和legend标签有两个作用。

（1）增强表单的语义。

（2）可以定义fieldset元素的disabled属性来禁用整个组中的表单元素。

对于图2-8中的效果，我们使用fieldset和legend这两个标签来增强语义化，修改后的代码如下。

```
<form action="index.aspx" method="post">
    <fieldset>
        <legend>登录绿叶学习网</legend>
        <p>
            <label for="name">账号：</label><input type="text" id="name" name
        </p>
```

```
        <p>
            <label for="pwd">密码：</label><input type="password" id="pwd" na
        </p>
        <input type="checkbox" id="remember-me" name="remember-me" /><label
        <input type="submit" value="登录" />
    </fieldset>
</form>
```

在浏览器预览效果如图2-10所示。



图2-10　加入fieldset和legend的表单

分析：

我们可以看到，使用了fieldset和legend这两个标签之后，表单形成了非常美观的“书签”效果。

2.6 其他语义化

2.6.1 换行符

很多新手会使用
标签来换行，或者使用多个
标签来实现元素之间的上下间距。

举例1：

```
<div>
    <span>标题</span><br/><br/>
    <span>第1部分内容</span><br/>
    <span>第2部分内容</span><br/>
    <span>第3部分内容</span>
</div>
```

举例2：

```
<form action="index.aspx" method="post">
    <fieldset>
        <legend>登录绿叶学习网</legend>
        <label for="name">账号：</label><input type="text" id="name" name="n
        <label for="pwd">密码：</label><input type="password" id="pwd" name=
        <input type="checkbox" id="remember-me" name="remember-me"/><label f
        <input type="submit" value="登录" />
    </fieldset>
</form>
```

上面两个例子使用
标签的方式是错误的，这也是
标签很常见的错误用法。事实上，
标签有自己特定的语义，不能随便用来实现换行效果。W3C标准规定，
标签仅仅用于段落中的换行，不能用于其他情况。也就是说，
标签只适合用于p标签内部的换

行，不能用于其他标签。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
</head>
<body>
    <p>广东省<br />广州市<br />黄埔大道西601号</p>
</body>
</html>
```

在浏览器预览效果如图2-11所示。

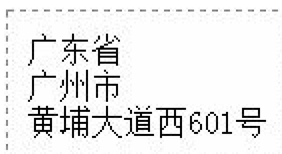


图2-11　
标签的正确用法

2.6.2　无序列表ul

在实际开发中，对于列表型的数据，为了实现良好的语义，我们还是建议使用无序列表或者有序列表，不建议使用div等来实现。

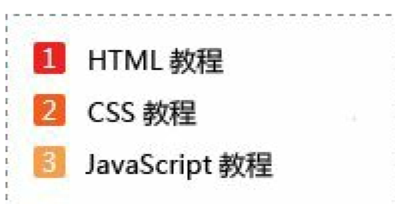


图2-12　列表效果

对于如图2-12所示效果，不少新手很可能会写出如下代码来实现。

```
<div>
```

```
    <div><span>1</span>HTML教程</div>
    <div><span>2</span>CSS教程</div>
    <div><span>3</span>JavaScript教程</div>
</div>
```

这种实现方式缺乏语义化，并且也不利于维护。正确的做法是：

```
<ul>
    <li><span>1</span>HTML教程</li>
    <li><span>2</span>CSS教程</li>
    <li><span>3</span>JavaScript教程</li>
</ul>
```

有人问，每一个列表项前都有数字，不应该使用有序列表来实现吗？为什么这里使用无序列表来实现呢？假如使用有序列表，我们是做不到这种外观效果的。因为有序列表前的数字外观是固定的。在实际开发中，大多数情况下都是使用无序列表，极少情况下会使用有序列表。



图2-13　百度的导航条



图2-14　绿叶学习网的图片列表

2.6.3　strong标签和em标签

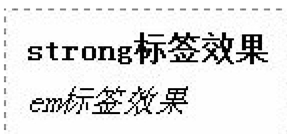


图2-15　strong和em标签效果

strong用于实现加粗文本，em用于实现斜体文本。基于结构和样式分离的原则，标签仅仅是为了实现简单的加粗或者斜体效果，我们一般不会用这两个。实际上，W3C对这两个标签赋予“强调”的语义，在strong或者em标签内部的文本被强调为重要文本。并且搜索引擎对这两个标签也赋予一定的权重。如果在一个页面中，为了SEO而想要突出某些关键字，可以使用strong和em这两个标签。一般情况下，我们都是去掉strong和em的默认样式，然后使用CSS重新定义新的样式，但这并不影响这两个标签的语义。也就是说，样式只会改变标签的外观，但不会改变标签的语义。对于去除标签的默认样式，我们在后面“4.5 CSS reset”一节会详细介绍。

2.6.4 del标签和ins标签

在HTML中，del和ins这两个标签是配合使用的。del表示“delete”，用于定义被删除的文本。ins表示“insert”，用于定义被更新的文本。一般情况下，我们会使用CSS来重新定义del和ins标签的样式。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
</head>
<body>
    <p>新鲜的新西兰奇异果</p>
    <p><del>原价:￥6.50/kg</del></p>
    <p><ins>现在仅售：￥4.00/kg</ins></p>
</body>
</html>
```

在浏览器预览效果如图2-16所示。

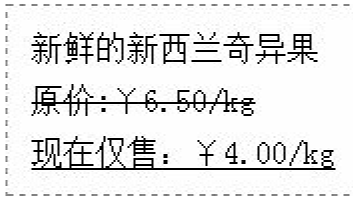


图2-16　del标签和ins标签效果

2.6.5　img标签

想要在页面显示一张图片，我们有两种方式：一是使用img标签；二是使用背景图片。这两种实现方式最明显的区别在于：使用img标签添加图片，是通过HTML来实现；使用背景图片，是通过CSS来实现。

在实际开发中，很多人添加图片的方式很随意。对于什么时候使用img标签，什么时候使用背景图片，并不是很清楚我们应该根据HTML的语义来判断。如果图片作为HTML的一部分，并且想要被搜索引擎识别，则应该使用img标签，例如常见的各种图片列表。如果图片仅仅是起到修饰作用，并且不想被搜索引擎识别，则应该使用背景图片。

举个例子，图2-17这个页面中的图标图片就应该使用背景图片实现，因为这些图标并不需要被搜索引擎识别，也不作为HTML的一部分。而图2-18页面中应该使用img标签来实现，因为这是页面HTML结构的一部分，并且希望被搜索引擎识别。



图2-17　背景图片实现的效果



图2-18　img标签实现的效果

【总结】

以上只是列举了在实际开发中比较常见的语义标签，其实HTML5新增了很多结构语义标签，例如header、nav、aside、footer、article、section等。如果想要实现语义更为良好的页面，我们也应该去关注这些新增的标签。不过结构语义标签是HTML5的内容，因此本书不展开介绍。

2.7　语义化验证

前面这几节，我们介绍了页面语义化需要注意的各个地方。那么平常有什么好的办法来判断一个页面是否语义良好呢？一个很简单的办法就是：去掉CSS样式，然后看页面是否还具有很好的可读性。

我们都知道，很多HTML标签都有一定的默认样式，例如p标签有上下边距、strong标签对字体加粗、ul标签有缩进效果，等等。

在前面我们接触过，我们可以使用一个标签来代替另外一个标签，并且使用CSS修饰来实现相同的效果。也就是说，不同的HTML标签可以通过不同的CSS来实现相同的效果。但是“一个语义良好的页面”跟“一个语义不好的页面”在去除样式之后的表现却是截然不同的。



图2-19　语义不好的页面去掉样式后的表现

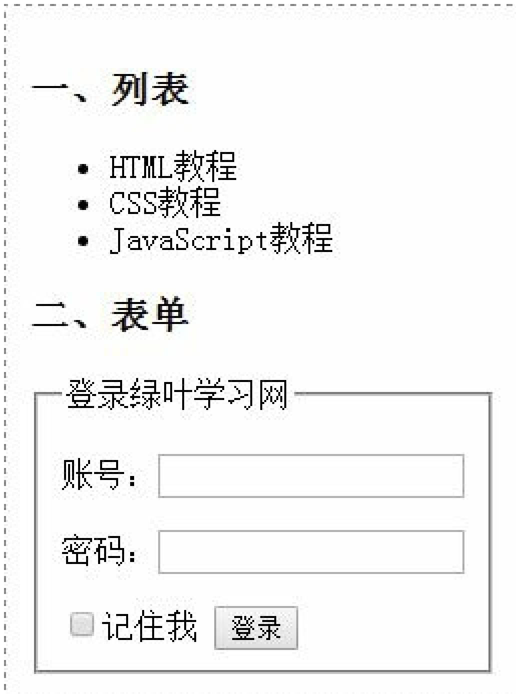


图2-20　语义良好的页面去掉样式后的表现

从上面两张图我们可以看出：一个语义良好的页面在“CSS裸奔”之后，可读性也是非常高的。想要查看一个页面在“CSS裸奔”下的效果，我们可以使用Firefox浏览器的一款网页调试插件“Web Developer”来测试。

在Web Developer工具栏找到“CSS”→“Disable Styles”→“Disable All Styles”并且选中，就可以查看页面去掉样式后的效果，如图2-21所示。Web Developer插件的安装和使用，请自行搜索，很简单。

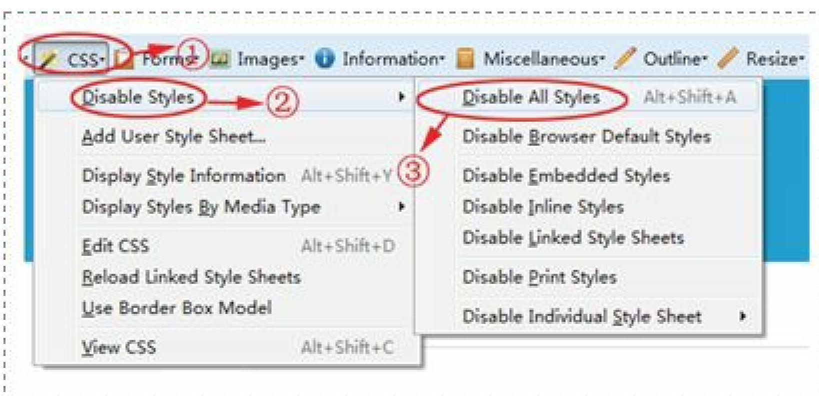


图2-21　Web Developer界面

2.8　HTML5舍弃的标签

在HTML5中，除了新增标签之外，也有不少标签被舍弃了，如表2-2和表2-3所示。为了实现页面的语义化，我们在实际开发中不应该再去使用这些标签。因此了解哪些标签已经被舍弃是非常有必要的。

对于这些被舍弃的标签，总体可以分为两大类。

（1）仅仅为了定义样式，没有任何语义，因此被舍弃。

（2）很少使用或者已经被新标签代替，因此被舍弃。

表**2-2 HTML5**舍弃的标签（仅为了定义样式）

标　　签	说　　明
basefont	定义页面文本的默认字体、颜色或尺寸
big	定义大字号文本
center	定义文本居中
font	定义文本的字体样式
strike	定义删除线文本
s	定义删除线文本
u	定义下划线文本

表**2-3 HTML5**舍弃的标签（很少使用或者已被新标签代替）

标　　签	说　　明
dir	定义目录列表，应该用ul代替
acronym	定义首字母缩写，应该用abbr代替
applet	定义嵌入的applet，应该用object代替
isindex	定义与文档相关的可搜索索引
frame	定义 frameset 中的一个特定的框架
frameset	定义一个框架集
noframes	为那些不支持框架的浏览器显示文本

对于HTML语义化，我们需要注意以下两点。

（1）我们应该果断舍弃那些仅仅为了定义样式而存在的HTML标签。如果仅仅是为了改变样式，我们应该使用CSS来实现，不要使用HTML标签。

（2）在不同的页面部分，我们优先使用正确的语义化标签。如果没有语义标签可用，才去考虑div和span等无语义标签。

第二部分　CSS进阶

第3章 CSS基础知识

3.1　CSS单位

在CSS入门阶段，我们大都习惯使用px作为单位。其实在CSS中，除了px，还有很多其他常用单位。总体来说，CSS单位分为两大类：绝对单位和相对单位。

3.1.1　绝对单位

在CSS中，绝对单位定义的大小是固定的，使用的是物理度量单位，显示效果不会受到外界因素影响，如表3-1所示。绝对单位一般多用于传统平面印刷，而极少用于前端开发。

表**3-1 CSS**绝对单位

绝 对 单 位	说　　明
cm	厘米
mm	毫米
in	英寸
pt	磅（point），印刷的点数
pc	pica，1pc=12pt

在前端开发中，我们都不会用到绝对单位，因此在这里只需要简单了解一下在CSS中有绝对单位这回事即可。

3.1.2　相对单位

在CSS中，相对单位定义的大小是不固定的，一般是相对其他长度而言。在CSS中，常用的相对单位如表3-2所示。

表**3-2 CSS**相对单位

绝 对 单 位	说　　明
px	像素
%	百分比
em	1em等于“当前元素”字体大小
rem	1rem等于“根元素”字体大小

除了上面这些，CSS相对单位还有ex、vw和vh等。这里我们只需要认真掌握上表中的单位，其他的相对单位很多都是用于移动端开发。在将来接触移动端开发时，我们再来认真学习。

1．px

px，全称pixel（像素，指的是一张图片中最小的点，或者是计算机屏幕中最小的点。

举个例子，图3-1是一个新浪图标。



图3-1　新浪图标

将这个图标放大n倍后效果如图3-2所示。

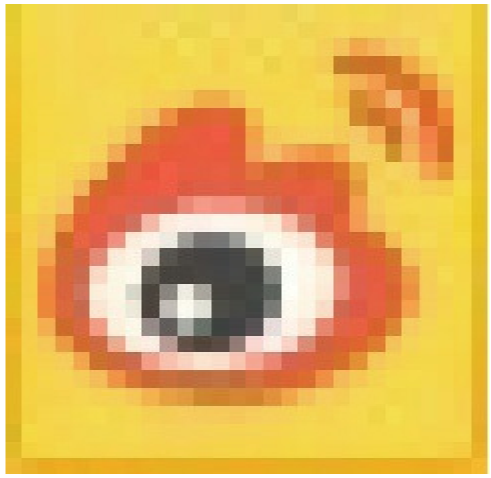


图3-2　放大后的新浪图标

我们发现，原来一张图片是由很多的小方点组成的。其中，每一个小方点就是一个像素（px）。

一台计算机的分辨率是800px×600px，指的就是“计算机显示屏宽是800个小方点，高是600个小方点”。

严格来说，px也是属于相对单位，因为屏幕分辨率的不同，1px的大小也是不同的。例如Windows系统的分辨率为每英寸96px，osx系统的分辨率为每英寸72px。如果不考虑屏幕分辨率，我们也可以把px当作绝对单位来看待，这也是为什么很多地方说px是绝对单位的原因。

2．%

在CSS中，支持百分比作为单位的属性很多，大致可以分为三大类。

（1）width、height、font-size的百分比是相对于父元素“相同属性”的值来计算的。

（2）line-height的百分比是相对于父元素的font-size值来计算的。

（3）vertical-align的百分比是相对当前元素的line-height值来计算的。

line-height和vertical-align这两个属性有点特殊，在后面章节会有详细介绍。拿font-size属性来说，如果父元素width为100px，子元素width为50%，则表示子元素实际width为50px。如果父元素font-size为30px，子元素font-size为50%，则表示子元素实际font-size为15px。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title></title>
     <style type="text/css">
        #father
        {
            width:200px;
            height:160px;
            border:1px solid blue;
            font-size:30px;
        }
        #son
        {
            width:50%;
            height:50%;
            border:1px solid red;
            font-size:50%;
        }
     </style>
</head>
<body>
     <div id="father">
         绿叶学习网
         <div id="son">绿叶学习网</div>
     </div>
</body>
</html>
```

在浏览器预览效果如图3-3所示。

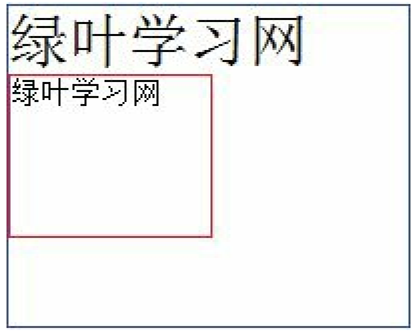


图3-3　百分比单位

3．em

在CSS中，em是相对于“当前元素”的字体大小而言的。其中，1em等于“当前元素”字体大小。这里的字体大小指的是以px为单位的font-size值。例如，当前元素的font-size值为10px，则1em等于10px；当前元素的font-size值为20px，则1em等于20px，以此类推。

此外需要注意一下，如果当前元素的font-size没有定义，则当前元素会继承父元素的font-size。如果当前元素的所有祖先元素都没有定义font-size，则当前元素会继承浏览器默认的font-size。其中，所有浏览器默认的font-size值都是16px。

在CSS中，使用em作为单位有以下三个小技巧。

（1）首行缩进使用text-indent:2em实现

我们都知道在网页排版中，段落的首行一般会缩进两个字的距离。如果要实现这个效果，text-indent值应该是font-size值的两倍。此时，我们使用“text-indent:2em”就可以轻松实现。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
```

```
    <style type="text/css">
        p
        {
            font-size:14px;
            text-indent:2em;
            width:360px;
        }
    </style>
</head>
<body>
     <h3>爱莲说</h3>
     <p>水陆草木之花，可爱者甚蕃。晋陶渊明独爱菊。自李唐来，世人甚爱牡丹。予独爱莲
     <p>予谓菊，花之隐逸者也；牡丹，花之富贵者也；莲，花之君子者也。噫！菊之爱，陶
</body>
</html>
```

在浏览器预览效果如图3-4所示。

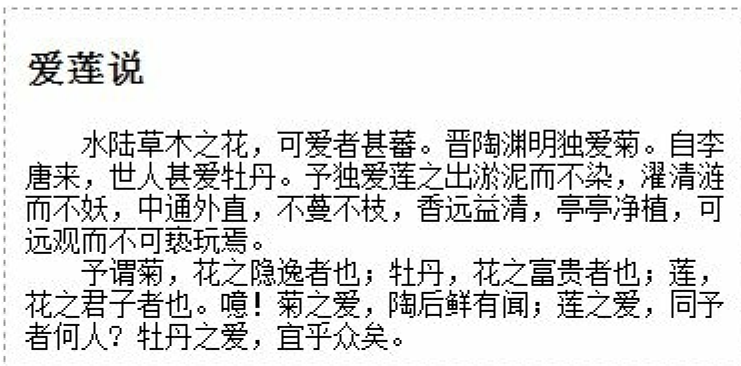


图3-4　em作为单位实例图（1）

分析：

在这个例子中，“text-indent:2em;”等价于“text-indent:28px;”。对于首行缩进，使用em作为单位会比使用px作为单位更好。为什么呢？因为使用px作为单位，如果我们定义font-size为14px，则text-indent就应该定义为28px；如果我们定义font-size为15px，则text-indent就应该定义为30px，以此类推……但是当我们使用em作为单位时，不管font-size定义为多少px，我们只需要将text-indent定义为2em即可。不需要计算，非常方便。

（2）使用em作为统一单位

首先，我们要非常清楚一点：所有浏览器默认字体大小都是16px。

如果在一个页面中，我们想要统一使用em作为单位，此时可以拿默认字体大小开刀。也就是说，对于任何元素我们都不需要设置font-size为多少px，而是继承根元素的font-size值（即16px），然后再使用em换算即可。

如果使用浏览器默认字体大小（16px），其中em与px对应关系如下。

```
1em = 16px×1 = 16px
0.75em = 16px×0.75 = 12px
```

为了简化font-size的计算，我们在CSS中提前声明“body{font-size:62.5%;}”。通过这个声明，我们可以使得默认字体大小变为16px×62.5%=10px，此时em与px对应关系如下。

```
1em = 10px
0.75em = 7.5px
```

也就是说，我们只需要将原来的px值除以10，然后换上em作为单位就行了。这是一个非常棒的技巧。在实际开发中，如果我们想要统一使用em作为单位，都是使用这个技巧。大家仔细琢磨以下两个实例，认真理解一下em的用法。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        body{font-size:62.5%;}
        #p1{font-size:1em;}
        #p2{font-size:1.5em;}
```

```
        #p3{font-size:2em;}
    </style>
</head>
<body>
    <p id="p1">当前字体大小为1em，也就是10px</p>
    <p id="p2">当前字体大小为1.5em，也就是15px</p>
    <p id="p3">当前字体大小为2em，也就是20px</p>
</body>
</html>
```

在浏览器预览效果如图3-5所示。

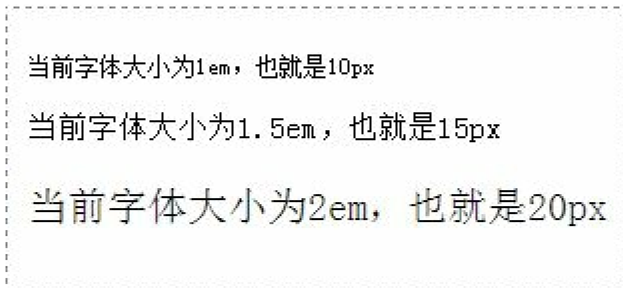


图3-5 em作为单位实例图（2）

一个p元素width为150px，height为75px，font-size为15px，text-indent为30px。如果我们想要全部使用em作为单位，该如何实现呢？请看下面的例子。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        *{padding:0;margin:0;}
        html{font-size:62.5%;}
        p{display:inline-block;border:1px solid silver;}
        /*使用px作为单位*/
        #p1
        {
            font-size:15px;
```

```
            width:150px;
            height:75px;
            text-indent:30px;
        }
        /*使用em作为单位*/
        #p2
        {
            font-size:1.5em;
            width:10em;
            height:5em;
            text-indent:2em;
        }
    </style>
</head>
<body>
    <p id="p1">绿叶学习网成立于2015年4月1日，是一个富有活力的技术学习网站。</p><
    <p id="p2">绿叶学习网成立于2015年4月1日，是一个富有活力的技术学习网站。</p>
</body>
</html>
```

在浏览器预览效果如图3-6所示。

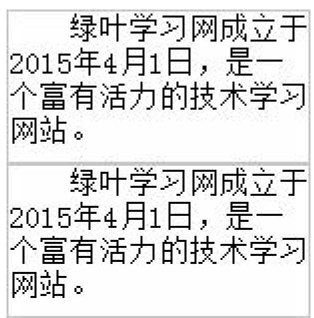


图3-6　em作为单位实例图（3）

分析：

在这个例子中，我们分别使用px和em作为单位，从而得出对比效果。有些小伙伴可能会产生疑惑：使用em作为单位时，width不应该是15em吗，为什么是10em呢？height不应该是7.5em吗，为什么是5em呢？

很多初学者都会有这样的疑问。不过很好解决，我们回过头来看看

em的定义：在CSS中，em是相对于“当前元素”的字体大小而言的。其中，1em等于“当前元素”字体大小。

稍微琢磨一下，大家都会明白为什么了。其实在这个例子中，当前元素的font-size为10px×1.5em=15px，如果width和height也要以em为单位，就要以当前元素的font-size值（15px）再计算一次。

```
width：150px/15px = 10em
height：75px/15px = 5em
```

也就是说，在实际开发中，对于em，我们一般需要计算两次：第一次，当前元素font-size属性的px值；第二次，当前元素其他属性（如width、height等）的px值。

（3）使用em作为字体大小单位

对于一个页面的字体大小，使用px作为单位时可扩展性不好，使用百分比作为单位时也不符合习惯，最佳选择是使用em作为单位来定义字体大小。使用em作为单位，当需要改变页面整体的文字大小时，我们只需要改变根元素字体大小即可，工作量变得非常少。em这个特点在跨平台网站开发中有着明显的优势。

4．rem

rem，全称font size of the root element，指的是相对于根元素（即html元素）的字体大小。rem是CSS3新引入的单位，目前除了IE8及以前版本之外，大部分主流浏览器都是支持rem的。rem布局是也是移动端最常用的字体大小之一。

我们可以通过这个网址查看rem的支持情况：

http://caniuse.com/#search=rem 。

rem跟em很相似，不过也有明显区别：em是相对“当前元素”的字体大小，而rem是相对“根元素”的字体大小。这里的font-size指的都是以px为单位的font-size值。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        html{font-size:62.5%;}
        #father
        {
            width:200px;
            height:160px;
            border:1px solid blue;
            font-size:2rem;
        }
        #son
        {
            width:150px;
            height:100px;
            border:1px solid red;
            font-size:2rem;
        }
    </style>
</head>
<body>
    <div id="father">
        绿叶学习网
        <div id="son">绿叶学习网</div>
    </div>
</body>
</html>
```

在浏览器预览效果如图3-7所示。

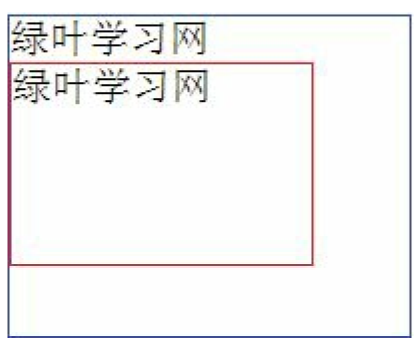


图3-7　rem作为单位

分析：

从这里我们可以看出，rem是相对根元素（html元素）的font-size而言的。

【疑问】

在实际开发中，CSS单位用px好，还是用em好呢？

在国外大部分主流网站都是使用em作为单位，而且W3C也建议我们使用em作为单位。但是我们发现国内大多数网站，包括三大门户“新浪、搜狐、网易”等，都是采用px作为单位。这是为什么呢？原来在国外，特别是美国，有一些法律规定网站要具有适应性。对于IE以前的版本无法调整那些使用px作为单位的字体大小，但现在的IE版本几乎都支持。我们也推荐读者使用px作为单位，因为用px作为单位非常方便计算长度。

3.2　CSS特性

CSS具有两大特性：继承性和层叠性。这里我们详细给大家介绍一下CSS的这两个特性。

3.2.1　继承性

CSS的继承性，指的是子元素继承了父元素的某些样式属性，例如在父元素定义字体颜色（color属性），子元素会继承父元素的字体颜色。不过我们要注意，并不是所有属性都具有继承性，如padding、margin、border等就不具备继承性。其实小伙伴们稍微想一下，要是padding、margin、border这些属性有继承性，那可是一件很恐怖的事情。

CSS的制定者（W3C组织）为我们考虑得很周到，只有那些能够给我们轻松书写的属性才可以继承。在CSS中，具有继承性的属性有三大类。

（1）文本相关属性：font-family、font-size、font-style、font-weight、font、line-height、text-align、text-indent、word-spacing。

（2）列表相关属性：list-style-image、list-style-position、list-style-type、list-style；

（3）颜色相关属性：color。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        #father{color:Red;font-weight:bold}
```

```
    </style>
</head>
<body>
    <div id="father">
        绿叶学习网
        <div id="son">绿叶学习网</div>
    </div>
</body>
</html>
```

在浏览器预览效果如图3-8所示。



图3-8　CSS继承性例子（1）

分析：

这里为父元素定义了color和font-weight两个属性，从预览效果中我们可以看到，子元素继承了父元素的这两个属性值。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
         #father{color:Red;font-weight:bold}
    </style>
</head>
<body>
    <div id="father">
         绿叶学习网
         <a href="http://www.lvyestudy.com">绿叶学习网</a>
```

```
    </div>
</body>
</html>
```

在浏览器预览效果如图3-9所示。



图3-9　CSS继承性例子（2）

分析：

咦，怎么回事？不是说color是继承属性吗？为什么在父元素定义“color:Red”，子元素（a元素）却没有变成红色呢？其实这是因为a标签本身有默认的颜色样式，优先级比继承的要高。如果想要改变a标签的颜色，必须选中a标签进行操作才行。对于a标签这个特点，我们在实际开发中会经常碰到。

利用CSS继承性，有时候我们可以少写很多代码。像上面这个例子，我们只需要在父元素中定义属性，就不需要在子元素中重复定义了。在实际开发中，我们应该充分利用CSS的继承性，从而减少重复代码的编写。这也使得我们的CSS代码更加精简优雅。

3.2.2　层叠性

在学习CSS层叠性之前，先问大家一个问题：如果在网页中，对于同一个元素，我们重复定义了多个相同的属性时，CSS会怎么处理呢？先看一个具体实例。

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        div{color:Red;}
        div{color:Orange;}
```

```
        div{color:Blue;}
    </style>
</head>
<body>
    <div>绿叶学习网</div>
    <div>绿叶学习网</div>
    <div>绿叶学习网</div>
</body>
</html>
```

在浏览器预览效果如图3-10所示。

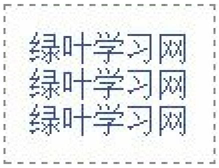


图3-10　CSS层叠性例子（1）

分析：

在这个例子中，我们首先定义了所有div颜色为Red，然后定义所有div颜色为Orange，最后定义所有div颜色为Blue。由于CSS的层叠性，“color:Orange;”会覆盖“color:Red;”，然后“color:Blue;”会覆盖“color:Orange;”。因此，最终所有div元素的颜色为Blue。

CSS的层叠性，指的就是样式的覆盖。对于同一个元素来说，如果我们重复定义多个相同的属性，并且这些样式具有相同的权重，CSS会以最后定义的属性的值为准；也就是“后来者居上”原则。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
```

```
    <style type="text/css">
        /*单独定义第2个div颜色为purple*/
        #second{color:Purple;}
        /*定义所有div颜色为red*/
        div{color:Red;}
    </style>
</head>
<body>
    <div>绿叶学习网</div>
    <div id="second">绿叶学习网</div>
    <div>绿叶学习网</div>
</body>
</html>
```

在浏览器预览效果如图3-11所示。

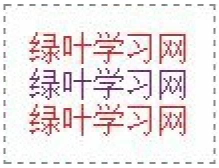


图3-11　CSS层叠性例子（2）

分析：

咦，这又是闹哪出？按“后来者居上”原则，所有div颜色最终都应该是Red，为什么第二个div颜色还是Purple呢。其实，“后来者居上”原则必须符合三个条件。

（1）元素相同。

（2）属性相同。

（3）权重相同。

权重，指的是选择器的权重。在接下来的“CSS优先级”一节中，我们再详细介绍。

3.3　CSS优先级

CSS，全称Cascading Style Sheet（层叠样式表）。很多人就只知道CSS是用来控制样式的，并没有深入理解“层叠”这两个字的含义。

层叠，其实指的就是样式的覆盖。当样式的覆盖发生冲突时，以优先级高的为准。当“同一个元素”的“同一个样式属性”被运用上多个属性值时，我们就需要遵循一定的优先级规则来选择属性值了。

样式覆盖发生冲突常见的共有以下五种情况。

（1）引用方式冲突。

（2）继承方式冲突。

（3）指定样式冲突。

（4）继承样式与指定样式冲突。

（5）!important。

3.3.1　引用方式冲突

我们知道CSS有三种常用的引入方式：外部样式、内部样式和行内样式。CSS引用方式的不同，也会产生冲突。这三种引入方式优先级如下。

行内样式>（内部样式 = 外部样式）

行内样式的优先级最高，内部样式与外部样式优先级相同。如果内部样式与外部样式同时存在，则以最后引入的样式为准（后来者居上）。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <link href="index.css" rel="stylesheet" type="text/css" />
    <style type="text/css">
        div{color:Orange;}
    </style>
</head>
<body>
    <div style="color:Blue;">绿叶学习网</div>
</body>
</html>
```

在浏览器预览效果如图3-12所示。



图3-12　引用方式冲突实例

分析：

在这个例子中，我们定义外部样式为“color:Red”，内部样式为“color:Orange;”，行内样式为“color:Blue;”。从优先级角度出发，“color:Orange;”会覆盖“color:Red;”，然后“color:Blue;”会覆盖“color:Orange”。因此字体最终颜色为Blue。

3.3.2　继承方式冲突

如果由于继承方式引起的冲突，则“最近的祖先元素”获胜。继承，指的是CSS的继承性。祖先元素，指的是当前元素的父元素、“爷”元素……当然，“爷”元素只是一个形象的叫法。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        body{color:Red;}
        #grandfather{color:Green;}
        #father{color:Blue;}
    </style>
</head>
<body>
    <div id="grandfather">
         <div id="father">
             <div id="son">绿叶学习网</div>
        </div>
    </div>
</body>
</html>
```

在浏览器预览效果如图3-13所示。



图3-13　继承方式冲突实例

分析：

由于继承方式引起的冲突，最近的祖先元素获胜，字体最终颜色为Blue。在这个例子中，#son元素的最近祖先元素为#father元素。如果#father元素没有定义color属性，则最近祖先元素为#grandfather元素。

3.3.3　指定样式冲突

所谓的指定样式，指的是指定“当前元素”的样式。当直接指定的样式发生冲突时，样式权值高者获胜。

在CSS中，各种选择器的权值如表3-3所示。

表**3-3** 选择器权值

选　择　器	权　　值
通配符	0
伪元素	1
元素选择器	1
class选择器	10
伪类	10
属性选择器	10
id选择器	100
行内样式	1000

常见的伪元素只有四个：即::before、::after、::first-letter、::first-line。伪类我们经常见到，如:hover、:first-child等。常用的选择器优先级如下。

行内样式> id选择器> class选择器>元素选择器

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
```

```
<head>
    <title></title>
    <style type="text/css">
        #lvye{color:Red;}
        .lvye{color:Green;}
        div{color:Blue;}
    </style>
</head>
<body>
    <div id="lvye" class="lvye">绿叶学习网</div>
</body>
</html>
```

在浏览器预览效果如图3-14所示。

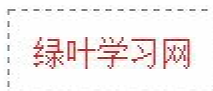


图3-14　指定样式冲突实例（1）

分析：

id选择器权重最高，因此div元素color属性最终值为Red。这里要注意一点，我们不应该只从样式顺序来判断。因为只有选择器权重相同时，才会遵从“后来者居上”原则。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        #outer p
        {
            /*权值=100+1=101*/
            color:Red;
```

```
        }
        #outer .inner
        {
            /*权值=100+10=110*/
            color:Green;
        }
        #outer p strong
        {
            /*权值=100+1+1=102*/
            color:Blue;
        }
        #outer p span strong
        {
            /*权值=100+1+1+1=103*/
            color:Purple;
        }
    </style>
</head>
<body>
    <div id="outer">
        <p class="inner">
            <span><strong>绿叶学习网</strong></span>
        </p>
    </div>
</body>
</html>
```

在浏览器预览效果如图3-15所示。

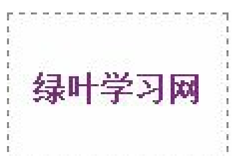


图3-15　指定样式冲突实例（2）

分析：

在预览效果中，我们可以看到strong标签的文本颜色为Purple（紫色）。怎么回事？按道理说，“#outer .inner{}”的权重最高，文本颜色应该为Green（绿色）啊。

其实之所以产生这种疑问，那是因为小伙伴们没有认真理解“指定样式冲突”是怎样一回事。所谓的指定样式冲突，指的是“当前元素”的样式发生冲突。

在这个例子中，我们所针对的当前元素是strong，然而“#outer .inner{}”针对的元素是p（strong的祖先元素），并不是strong。

准确来说，如果当前元素为strong，则“#outer .inner{}”和“#outer p”都属于继承样式。在继承样式中，我们是不能用选择器权重这一套东西来计算的。

由此我们可以总结：在CSS中，选择器权重的计算只针对指定样式（当前元素），并不能用于继承样式。

3.3.4 继承样式和指定样式冲突

当继承样式和指定样式发生冲突时，指定样式获胜。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        body{color:Red;}
        #outer{color:Green;}
        #outer #inner{color:Blue;}
        span{color:Purple;}
        strong{color:Black;}
    </style>
</head>
<body>
    <div id="outer">
        <p class="inner">
            <span><strong>绿叶学习网</strong></span>
        </p>
    </div>
</body>
```

```
</html>
```

在浏览器预览效果如图3-16所示。



图3-16　继承样式和指定样式冲突实例

分析：

由于CSS的继承性，strong元素分别从body、div和p这三个元素继承了color属性，但是这些都属于继承样式。最后，由于“strong{color:Black;}”这一句指定了strong元素的color属性（指定样式），因此最终strong元素color属性为Black。

对于样式冲突，我们不能笼统地用权重值来计算，例如“body{color:Red;}”权重值为1，“#outer{color:Green;}”为100，“#outer #inner{color:Blue;}”权重值为200，“span{color:Purple;}”权重值为1，“strong{color:Black;}”权重值为1，然后就判断strong元素color属性为Blue。

在CSS中，选择器权重值的计算只针对指定样式（当前元素），并不能用于继承样式。当继承样式和指定样式发生冲突时，指定样式获胜。我们先判断指定样式，然后才考虑继承样式。

3.3.5　!important

在CSS中，我们可以使用!important规则来改变样式的优先级。如果一个样式使用!important来声明，则这个样式会覆盖CSS中任何的其他样式声明。也就是说，如果你一定要使用某个样式属性，为了不让它被覆盖，则可以使用!important来实现。换句话说，如果你想要覆盖其他所有样式，可以使用!important来实现。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        #outer strong
        {
            /*权重=100+1=101*/
            color:Red;
        }
        #outer .inner strong
        {
            /*权重=100+10+1=111*/
            color:Green;
        }
        .inner strong
        {
            /*权重=10+1=11*/
            color:Blue;
        }
        .inner span strong
        {
            /*权重=10+1+1=12*/
            color:Purple;
        }
        strong
        {
            color:Black !important;
        }
    </style>
</head>
<body>
    <div id="outer">
        <p class="inner">
            <span><strong>绿叶学习网</strong></span>
        </p>
    </div>
</body>
</html>
```

在浏览器预览效果如图3-17所示。



图3-17　!important实例

分析：

在预览效果中，strong元素color属性值为Black。按正常来说，“#outer .inner strong{}”权值最高，strong元素color属性值应该是Green。但是由于“strong{}”中应用了!important，因此strong元素color属性值最终为Black。

1．!important用处

在CSS中，!important常见有两种使用情况。

（1）情况一

```
#someId p{color:Red;}
p{color:Green;}
```

在外层有#someId的情况下，你怎样才能让p变为Green呢？在这种情况下，如果不使用!important，则第一条样式永远比第二条样式优先级更高。

（2）情况二

我们可能会碰到这种情况：你或者同事写了一些效果很差的行内样式（行内样式优先级往往是最高的），假如你要在内部样式表或者外部样式表中修改这个样式，你就应该想到在内部样式表或者外部样式表中使用!important来覆盖那些行内样式。

举一个活生生的例子：有人在jQuery插件中写了糟糕的行内样式，而你需要在CSS文件中修改这些样式。

2. 如何覆盖!important

想要覆盖!important声明的样式很简单，共有两种解决方法。

（1）使用相同的选择器，再添加一条!important的CSS语句。

不过这个CSS语句得放在后面。因为在优先级相同的情况下，后面定义的会覆盖前面定义的（后来者居上）。

（2）使用更高优先级的选择器，再添加一条!important的CSS语句。

使用“大杀器”!important可以将样式提升到最高等级，不管这个样式在哪个样式表还是在哪个选择器中。如果在同一样式中出现了多个!important，则遵循后来者居上原则。

在实际开发的过程中，经常会碰到写在后面的样式被写在前面的样式覆盖了，这时候就应该考虑一下是否是CSS优先级引起的问题。了解CSS优先级的这些规则，能为我们提高不少开发效率。对于样式冲突这五种情况，我们只需要认真地把每一个规则都理解一遍即可，并不需要记住。经过一定的实战，我们自然而然会有深刻的理解。

总而言之，对于CSS优先级，主要就是以下两个黄金定律。

（1）优先级高的样式覆盖优先级低的样式。

（2）同一优先级的样式，后定义的覆盖先定义的，即后来者居上。

3.4　CSS引入方式

在初学者阶段，我们接触了三种CSS引入方式，我们来简单回顾一下。

（1）外部样式表。

（2）内部样式表。

（3）行内样式表。

除了这三种方式，在CSS中其实还有一种@import方式（即“导入样式表”）。@import方式跟外部样式表很相似。不过在实际开发中，我们极少使用@import方式，而更倾向于使用link方式（外部样式）。原因在于@import方式先加载HTML后加载CSS，而link先加载CSS后加载HTML。如果HTML在CSS之前加载，页面用户体验非常差。因此，我们不需要去了解@import方式。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <!--这是外部样式，CSS样式在外部文件中定义-->
    <link href="index.css" rel="stylesheet" type="text/css" />
    <!--这是内部样式，CSS样式在style标签中定义-->
    <style type="text/css">
        p{color:Red;}
    </style>
</head>
<body>
    <!--这是行内样式，CSS样式在元素style属性中定义-->
    <p style="color:Blue;">绿叶学习网</p>
    <p>绿叶学习网</p>
    <p>绿叶学习网</p>
```

```
</body>
</html>
```

3.4.1　外部样式表

外部样式表是最理想的CSS引用方式。在实际开发当中，为了提升网站性能和可维护性，一般都是使用外部样式表。所谓的“外部样式表”，就是把CSS代码和HTML代码单独放在不同文件中，然后在HTML文档中使用link标签来引用CSS样式表。

当样式需要被应用到多个页面时，外部样式表是最理想的选择。使用样式表，就可以通过更改一个CSS文件来改变整个网站的外观。

外部样式表在单独文件中定义，并且在<head></head>标签对中使用link标签来引用。

3.4.2　内部样式表

我们都知道外部样式表是最理想的CSS引用方式，那是不是意味着内部样式表和行内样式表这两种方式就一无是处呢？肯定不是。

在实际开发中，相同频道的页面都会有相同的样式。对于这种公有样式，我们一般会放在外部样式表中。例如绿叶学习网的所有文章页面都同属一个频道，这些页面都具有相同的样式（假如CSS文件为article.css），如图3-18所示。



3.5 字体粗细font-weight

作者(helicopter)　赞(10)　浏览(6336)　评论(7)　说明:原创教程，禁止转载

一、font-weight属性

在CSS中，使用font-weight属性来定义字体粗细。

初学者要注意，字体粗细和字体大小（font-size）是不一样的，粗细指的是字体的“肥胖”，大小指的是高和宽。大家好好理解一下就很容易区分了。

语法：

font-weight:粗细值;

说明：

粗细值可以取关键字和100~900的数值。关键字如下表。

表1 font-weight属性

font-weight属性值	说明
normal	默认值，正常体
lighter	较细
bold	较粗
bolder	很粗（其实效果跟bold差不多）

字体粗细font-weight属性值可以取100、200、…、900这九个值。400相当于正常字体normal，700相当于bold。100~900分别表示字体的粗细，是对字体粗细的一种量化方式，值越大就表示越粗，值越小就表示越细。

图3-18　绿叶学习网的文章页面

当有一些页面需要“个别样式”时，此时我们就不应该把这些“个别样式”放在“公有样式”（article.css）中去。因为这些个别样式只应用在几个页面，如果我们把这些个别样式放到公有样式中，会导致所有页面都加载一次个别样式，这样会影响加载速度。遇到这种情况时，我们就不能单纯只用外部样式表来解决，比较好的解决方法是：在这些需要定义个别样式的页面中使用内部样式表来定义。

在绿叶学习网文章页面中，大部分页面的表格都是两列的，因此我们把两列表格样式放入公有样式（article.css）中。但是有极个别页面需要用到三列表格，如图3-19所示。此时，我们在需要用到的页面中的内部样式表中定义即可。

transform-origin属性取值

关键字	百分比	说明
top left	0 0	左上
top center	50% 0	靠上居中
top right	100% 0	右上
left center	0 50%	靠左居中
center center	50% 50%	正中
right center	100% 50%	靠右居中
bottom left	0 100%	左下
bottom center	50% 100%	靠下居中
bottom right	100% 100%	右下

图3-19　绿叶学习网文章页面的三列表格

3.4.3　行内样式表

在一个样式非常多的页面里，我们只在一个小地方进行样式修改（如加粗、改变颜色等）。像这种样式修改只出现一两次，并且修改幅度小。此时，我们更倾向于使用行内样式来实现。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
</head>
<body>
    <div><span style="font-weight:bold;color:red;">绿叶学习网</span>成立于201
</body>
</html>
```

在浏览器预览效果如图3-20所示。

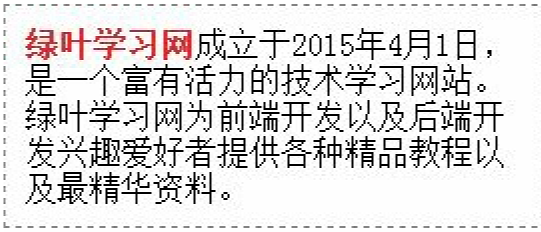


图3-20　行内样式实例

分析：

在这个例子中，我们要实现对“绿叶学习网”这个词进行加粗并且改变颜色。如果这个页面内容过多、CSS样式过大的话，使用行内样式中实现会更加方便。首先，外部样式我们是不可能考虑的。其次，如果使用内部样式表，我们可能还得为这个span元素定义一个id或者class，这显得十分多余。因此最好的办法还是使用行内样式。这个技巧在我们官网也经常用到如图3-21所示。



图3-21　绿叶学习网使用的行内样式

【总结】

在实际开发中，我们应该灵活地配合使用外部样式表、内部样式表以及行内样式表，并不是一味地只用外部样式表。事实上，外部样式表多用于公有样式，内部样式表多用于私有样式，而行内样式更多用于小修改或者优先级方面。

3.5　CSS选择器

选择器，说白了就是用一种方式把你想要的那一个标签选中。把它选中了，你才能操作这个标签的CSS样式。CSS中有很多把你所需要的标签选中的方式，这些不同的方式就是不同的选择器。

在CSS入门阶段，我们学习了以下几种选择器。

（1）元素选择器

（2）id选择器

（3）class选择器

（4）群组选择器

这些都是CSS中最基本的选择器，大家可以到绿叶学习网的CSS入门教程中复习一下，在这里就不再详细展开。在这一节中，我们给大家详细讲解CSS2.1中的层次选择器。

层次选择器，就是通过元素之间的层次关系来选择元素。层次选择器在实际开发中也是相当重要的。常见的层次关系包括：父子、后代、兄弟和相邻。

在CSS中，层次选择器共有四种，如表3-4所示。

表**3-4 CSS**层次选择器

选　择　器	说　　明
M　N	后代选择器，选择M元素内部后代N元素（所有N元素）
M>N	子代选择器，选择M元素内部子代N元素（所有第1级N元素）
M~N	兄弟选择器，选择M元素后所有的同级N元素

M+N	相邻选择器，选择M元素相邻的下一个元素（M、N是同级元素）

3.5.1 后代选择器

后代选择器，就是选择元素内部中所有的某一个元素，包括子元素和其他后代元素。

语法：

```
M N{}
```

说明：

在后代选择器中，“M元素”和“N元素”之间用空格隔开，表示选中M元素内部后代N元素（所有N元素）。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        #firstp{color:Red;}
    </style>
</head>
<body>
    <div id="first">
        <p>lvye的子元素</p>
```

```
        <p>lvye的子元素</p>
        <div id="second">
            <p>lvye子元素的子元素</p>
            <p>lvye子元素的子元素</p>
        </div>
        <p>lvye的子元素</p>
        <p>lvye的子元素</p>
    </div>
</body>
</html>
```

在浏览器预览效果如图3-22所示。

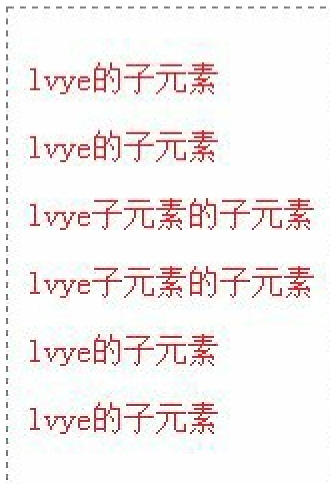


图3-22　后代选择器

分析：

“#first p”表示选取id为first的元素内部的所有p元素，因此不管是子元素，还是其他后代元素，全部都会被选中。

3.5.2　子代选择器

子代选择器，就是选中元素内部的某一个子元素。子代选择器跟后代选择器很相似，但是却有着明显的区别。

（1）后代选择器，选取的是元素内部所有的元素（包括子元

素）。

（2）子代选择器，选取的是元素内部某一个子元素（只限子元素）。

语法：

```
M>N{}
```

说明：

在子代选择器中，“M元素”和“N元素”之间使用“>”选择符，表示选中M元素内部的子元素N。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        #first>p{color:Red;}
    </style>
</head>
<body>
    <div id="first">
        <p>lvye的子元素</p>
        <p>lvye的子元素</p>
        <div id="second">
            <p>lvye子元素的子元素</p>
            <p>lvye子元素的子元素</p>
        </div>
        <p>lvye的子元素</p>
```

```
        <p>lvye的子元素</p>
    </div>
</body>
</html>
```

在浏览器预览效果如图3-23所示。

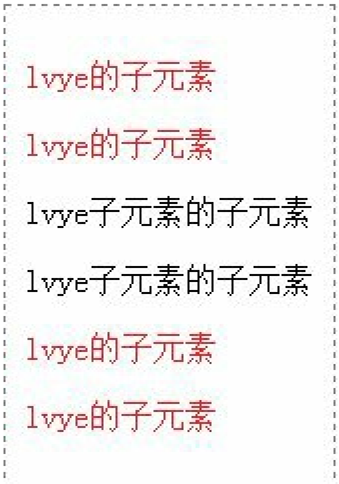


图3-23　子代选择器

分析：

“#first>p”表示选中id为first的元素下的子元素p。这里与后代选择器的例子对比一下，我们很清楚地知道：子代选择器只选取子元素，不包括其他后代元素。

3.5.3　兄弟选择器

兄弟选择器，就是选中元素后面（不包括前面）的某一类兄弟元素。

语法：

```
M~N{}
```

说明：

在兄弟选择器中，“M元素”和“N元素”之间使用“～”选择符，表示选中M元素后面的所有某一类兄弟元素N。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        #second~p{color:Red;}
    </style>
</head>
<body>
    <div id="first">
        <p>lvye的子元素</p>
        <p>lvye的子元素</p>
        <div id="second">
            <p>lvye子元素的子元素</p>
            <p>lvye子元素的子元素</p>
        </div>
        <p>lvye的子元素</p>
        <p>lvye的子元素</p>
    </div>
</body>
</html>
```

在浏览器预览效果如图3-24所示。

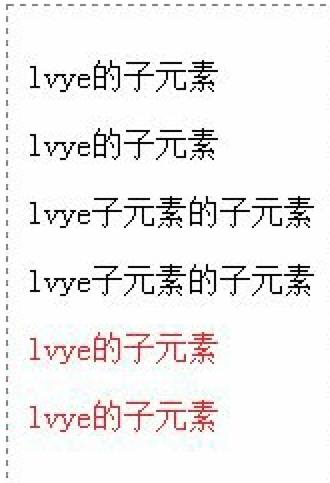


图3-24　兄弟选择器

分析：

“#second~p”表示选取id为second的元素后面的所有兄弟元素p。记住，兄弟选择器只选取后面的所有兄弟元素，不包括前面的所有兄弟元素。

3.5.4　相邻选择器

相邻选择器，就是选中元素后面（不包括前面）的某一个“相邻”的兄弟元素。相邻选择器跟兄弟选择器也非常相似，不过也有明显的区别。

（1）兄弟选择器选取元素后面“所有”的某一类元素。

（2）相邻选择器选取元素后面“相邻”的某一个元素。

语法：

```
M+N{}
```

说明：

在相邻选择器中，“M元素”和“N元素”之间使用“+”选择符，表示选中M元素后面的某一个相邻的兄弟元素N。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title></title>
    <style type="text/css">
        #second+p{color:Red;}
    </style>
</head>
<body>
    <div id="first">
        <p>lvye的子元素</p>
        <p>lvye的子元素</p>
        <div id="second">
            <p>lvye子元素的子元素</p>
            <p>lvye子元素的子元素</p>
        </div>
        <p>lvye的子元素</p>
        <p>lvye的子元素</p>
    </div>
</body>
</html>
```

在浏览器预览效果如图3-25所示。

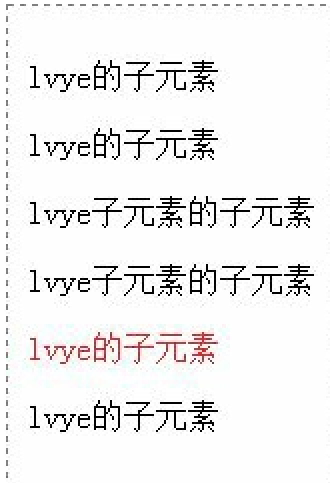


图3-25　相邻选择器（1）

分析：

“#second+p”表示选取id为second的元素后面的“相邻”的兄弟元素p。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        /*去除所有元素默认的padding和margin*/
        *{padding:0;margin:0}
        /*去除列表项默认符号*/
        ul{list-style-type:none;}
        li+li{border-top:2px solid red;}
    </style>
</head>
<body>
    <ul>
        <li>第1个元素</li>
        <li>第2个元素</li>
```

```
        <li>第3个元素</li>
        <li>第4个元素</li>
        <li>第5个元素</li>
        <li>第6个元素</li>
    </ul>
</body>
</html>
```

在浏览器预览效果如图3-26所示。



图3-26　相邻选择器（2）

分析：

“li+li”使用的是相邻选择器，表示“选择li元素相邻的下一个li元素”。由于最后一个li元素没有相邻的下一个li元素，所以对于最后一个li元素，它是没有下一个li元素可以选取的。“li+li{border-top:2px solid red;}”达到在两个li元素之间添加一个边框的效果。

这是一个非常棒的技巧，在实际开发中如果我们想要在两个元素之间实现什么效果（border、margin等），我们会经常用到这个技巧！大家一定要重点掌握。

例如图3-27所示的底部信息栏就可以用这个技巧来实现，大家可以尝试自己实现一下。

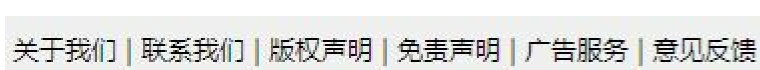


图3-27　绿叶学习网底部信息栏

在这一节，其实我们主要讲解的是两组选择器。

（1）“后代选择器”和“子代选择器”。

（2）“兄弟选择器”和“相邻选择器”。

这样划分就一目了然了。大家可以根据这个划分，深入对比，更能加深理解和记忆。

至此，我们已经把CSS2.1中的选择器学得差不多了。除了这些，其实在CSS中，还有非常多重要的选择器，不过大部分都是CSS3新增加的。本书只是重点讲解CSS2.1的开发技巧，对于CSS3的内容就不再展开介绍。不过作为过来人，还是给小伙伴们一个温馨建议：CSS3实在太强大了，能够实现很多非常棒的效果，学了本书内容之后一定要去学习一下CSS3。

第4章　CSS规范

4.1　CSS规范简介

作为前端开发人员，先问问小伙伴们在写CSS时是否碰到以下问题。

（1）你总是看不懂别人写的代码，或者读起来很吃力?

（2）你总是怕自己编写的代码与同事的有冲突或者互相影响?

（3）你的代码在多次维护之后，是否变得越来越臃肿，越来越难以维护?

（4）当你需要修改同事写的代码时感觉无从下手，或者要去问他如果改了这里那里会不会影响其他代码？

……

其实出现上面这些情况的根本原因在于，CSS代码没有规范化！

规范化的CSS不仅利于团队合作，而且对后期的维护以及代码的重用，都非常重要。在这一章里面，我们从以下四个方面来介绍一下关于CSS规范的内容。

（1）命名规范。

（2）书写规范。

（3）注释规范。

（4）CSS reset。

4.2　命名规范

不少人包括那些使用了CSS很长时间的开发人员，面对CSS文件的命名或者元素id以及class的命名都会犯愁或者很随意地对待。其实，一个良好的命名规范，不仅可以提高代码的阅读体验，而且这对搜索引擎优化也是非常重要的。

对于命名规范，主要有两个方面：CSS文件命名、id和class命名。

4.2.1　CSS文件命名

CSS文件命名及说明如表4-1所示。

表**4-1 CSS**文件命名

文　件　名	说　　明
reset.css	重置样式，重置元素默认样式
global.css	全局样式，全站公用，定义页面基础样式
themes.css	主题样式，用于实现换肤功能
module.css	模块样式，用于模块的样式
master.css	母版样式，用于母版页的样式
columns.css	专栏样式，用于专栏的样式
forms.css	表单样式，用于表单的样式

mend.css	补丁样式，用于维护、修改的样式
print.css	打印样式，用于打印的样式

reset.css用于去除元素的默认样式，使得页面在所有浏览器中有统一的外观。关于重置样式，我们在4.5节“CSS reset”中会介绍。

global.css用于定义公共模块样式以及基础样式。常见的公共模块包括导航、底部信息栏等。基础样式包括全局字体、文字颜色等。

那么最大的疑问来了！平常我们都是把重置样式、导航样式全都写在一个CSS文件里面，这里为什么还要搞那么多个文件出来呢？再说了，一个页面引入这么多CSS文件， 会引发多次HTTP请求，页面加载速度岂不是非常慢?

事实上，把样式文件划分为多个文件，这是“开发阶段”的做法！按照功能模块划分CSS文件，那是为了方便在开发阶段进行开发和修改。在整站发布的时候，我们会使用工具将多个CSS文件压缩合并成一个文件。也就是说，开发阶段和发布阶段是有区别的，大家要分清楚。

4.2.2　id和class命名

不少新手朋友对待元素id和class的使用和命名很随意，似乎完全是凭心情来的。例如这个元素用了id，那个元素就用用class吧。其实，什么时候用id，什么时候用class，是很讲究的。对于id和class的使用，我们在HTML进阶部分已经给大家详细讲解过了。

此外，对于元素id和class的命名，其实也是很有讲究的。好的命名有很多优点，不仅方便阅读、方便维护，而且对搜索引擎优化也是相当重要的。搜索引擎识别一个页面，很大一部分是根据元素id或class命名来判断。假如一个页面的元素命名很随意的话，搜索引擎小蜘蛛很容易迷路，这样会导致它以后很少来光顾你的网站。

id和class命名，比较常见的有三种方法。

（1）骆驼峰命名法，例如topMain、subLeftMenu。

（2）中划线命名法，例如top-main、sub-left-menu。

（3）下划线命名法，例如top_main、sub_left_menu。

在CSS中，对于元素id和class的命名，我们给出几点建议。

（1）使用英文命名而非中文命名。例如页面头部应该命名为header，而不是toubu。

（2）尽量不缩写，除非是一看就明白的单词，例如nav。

（3）命名统一规范，尽量不要出现一个用中划线命名，而另外一个用下划线命名的情况。虽然下划线和中划线都可以，但是建议使用中划线，可参见各大型网站。

（4）为了避免class命名的重复，命名时一般取父元素的class名作为前缀，例如：

```
<div class="column">
    <h3 class="column-title"></h3>
    <div class="column-content"></div>
</div>
```

1. 网页主体部分命名

表4-2所示的是页面中常见部分的命名规范建议。

表**4-2**　网页主题部分命名

网 页 部 分	命　　名
最外层	wrapper（一般用于包裹整个页面）
头部	header
内容	content

侧栏	sidebar
栏目	column
热点	hot
新闻	news
下载	download
标志	logo
导航条	nav
主体	main
左侧	main-left
右侧	main-right
底部	footer
友情链接	friendlink
加入我们	joinus
版权	copyright
服务	service
登录	login

注册	register

2. 网页细节部分命名

（1）导航（见表4-3）。

表**4-3**　导航命名

网 页 部 分	命　　名
主导航	main-nav
子导航	sub-nav
边导航	side-nav
左导航	leftside-nav
右导航	rightside-nav
顶导航	top-nav

（2）菜单（见表4-4）。

表**4-4**　菜单命名

网 页 部分	命　　名
菜单	menu

子菜单	submenu

（3）其他（见表4-5）。

表**4-5** 其他命名

网 页 部 分	命　　名
标题	title
摘要	summary
登录条	loginbar
搜索	search
标签页	tab
广告	banner
小技巧	tips
图标	icon
法律声明	siteinfolegal
网站地图	sitemap
工具条	tool、toolbar
首页	homepage

子页	subpage
合作伙伴	parter
帮助	help
指南	guide
滚动	scroll
提示信息	msg
投票	vote
相关文章	related
文章列表	list

对于上面这些命名规范，建议小伙伴们在实际开发中多多参考，使得自己编写的代码更规范，更具有语义性和可读性，方便后期维护。

4.3 书写规范

在CSS中，属性的书写顺序也是很有讲究的。良好的书写顺序习惯，既方便阅读，同时也方便后期维护。Andy Ford和Fantasai这两位都是CSS领域内的专家，他们都各自对CSS属性书写顺序提出过自己的意见。表4-6所示的是综合两位专家的思想所推荐的CSS属性书写顺序。

表**4-6 CSS**属性书写顺序

属 性 类 别	举　　例
影响文档流属性（布局属性）	display、position、float、clear等
自身盒模型属性	width、height、padding、margin、border、overflow等
文本性属性	font、line-height、text-align、text-indent、vertical-align等
装饰性属性	color、background-color、opacity、cursor等
其他属性	content、list-style、quotes等

这种CSS属性书写顺序是按照样式功能的重要性从上到下排列的。这里把影响元素页面布局的样式（如float、margin、padding、height、width等）排到前面，而把不影响布局的样式（如background、color、font等）排到后面。这种主次分明的排列方式，极大地提高了代码的可阅读性和可维护性。

举例：

```
#main
{
    /*影响文档流属性*/
    display:inline-block;
    position:absolute;
    top:0;
    left:0;
    /*盒子模型属性*/
    width:100px;
    height:100px;
    border:1px solid gray;
    /*文本性属性*/
    font-size:15px;
    font-weight:bold;
    text-indent:2em;
    /*装饰性属性*/
    color:White;
    background-color:Red;
    /*其他属性*/
    cursor:pointer;
}
```

上面CSS书写顺序是比较规范的，读起来一目了然，后期维护也很方便。对于这种书写顺序，一开始我们并不适应。我们应该在实际开发的过程中感性地认知，并逐步规范自己的书写顺序。

在这里，大家可能就有疑问了：在实际开发中，是不是一定要把影响文档流的属性写完了，然后才去写盒子模型的属性；是不是一定要把文本性属性写完了，然后才去写装饰性属性呢？

这倒完全没必要，而且我们也做不到。因为CSS中的属性是随着开发的需要逐步添加的。也就是说对于属性书写顺序，我们只关心“书写结果”，并不关心“书写过程”。 我们只需要保证最终的CSS代码顺序符合规范就可以了。因为这样可以方便我们阅读以及维护。

举个例子，一开始我们可能写下以下代码：

```
#main
{
    /*盒子模型属性*/
    width:100px;
    height:100px;
    border:1px solid gray;
```

```
}
```

然后写着写着，可能我们会发现需要添加定位属性，这时候就应该在盒子模型属性前面加入如下代码。

```
#main
{
    /*影响文档流属性*/
    display:inline-block;
    position:absolute;
    top:0;
    left:0;
    /*盒子模型属性*/
    width:100px;
    height:100px;
    border:1px solid gray;
}
```

在实际开发中为了更好地规范CSS书写顺序，我们还得分两个方面来考虑：普通代码和功能代码。

1. 普通代码

对于一般情况，我们应该保证CSS代码的最终顺序按照表4-6中的书写顺序。

2. 功能代码

对于单行文本居中、块元素居中等具有某一个特殊功能的代码块，我们就不应该那么呆板了。因为功能代码往往是用多个CSS属性来实现的，此时如果也按照表4-6的书写顺序来的话，这些功能代码就会被打乱，并且难以维护。因此，对于功能代码，我们应该集中放在一块。

举例：

```
#main
{
```

```
    float:left;
    width:100px;
    /*单行文本居中*/
    height:50px;
    line-height:50px;
    border:1px solid gray;
    font-size:15px;
    color:White;
}
```

对于初学者来说，这些功能代码也应该加入注释，以便阅读时一目了然。

4.4 注释规范

在CSS中，为一些关键代码做一下注释是非常必要的。注释的好处很多，比如方便理解、方便查找或方便项目组里的其他开发人员了解你的代码，而且可以方便以后你对自己的代码进行修改。

此外，良好的注释规范对于可读性也是非常重要的。下面从几个方面给大家一些关于CSS注释规范方面的建议。

4.4.1 顶部注释

顶部注释是文件的基本信息，一般包括文件说明、文件版本（更新）、作者以及版权声明等。

举例：

```
/*
 *@description:说明
 *@author:作者
 *@update:更新时间，如2016-4-19 18:30
 */
```

4.4.2 模块注释

模块注释是各个模块（如导航、底部信息栏等）的注释说明，模块注释建议要说明“开始”和“结束”，以便一目了然。

举例：

```
/*导航部分，开始*/
```

```
……
/*导航部分，结束*/
```

4.4.3　简单注释

简单注释一般用于注释某些关键代码，如功能代码。简单注释分为单行注释和多行注释这两种。

单行注释如下：

```
/*单行注释内容*/
```

多行注释如下：

```
/*
 *多行注释内容
 *多行注释内容
 *多行注释内容
 */
```

上面这些都是一些比较好的注释规范的建议。那么有人就问了，在网站发布的时候我们往往都需要使用压缩工具对CSS文件进行压缩。压缩之后这些注释不是去掉了吗？为什么我们还要那么费心费力地去注释呢？

其实这并不矛盾，我们做好注释是为了方便开发以及后期的维护。在整站发布的时候，我们会使用工具进行压缩才发布出去。开发阶段和发布阶段是有区别的，大家要分清楚。

我们都知道压缩工具会删除所有的注释，有些时候为了保留一些版权声明的注释说明，可以采用以下方式：

```
/*!注释内容*/
```

也就是说在注释内容最前面加上一个“!”，这样压缩工具就不会删除这条注释信息了。

此外还需要特别注意一下，CSS注释的方式都是“/注释内容/”，而不包括“//注释内容”这种方式。不过编辑器会给出正确的提示，这个我

们也不容易出错。对于CSS工具，我们在12.4节“压缩工具”中会详细介绍。

4.5 CSS reset

4.5.1 什么是CSS reset

我们都知道，在HTML中很多元素都有一定的默认样式。表4-7所示，列举了一些HTML中常见元素默认样式。

表4-7 HTML元素的默认样式

元　　素	默 认 样 式
p	有上下边距
strong	有字体加粗样式
em	有字体倾斜样式
ul	有缩进样式

reset就是“重置”的意思，CSS reset指的就是重置样式。所谓的重置样式，最简单的说法就是：去除元素在浏览器的默认样式。

4.5.2 为什么要用CSS reset

现在浏览器有很多，主流的有Chrome、Firefox、IE、Safari、Opera等。不同的浏览器，默认样式也是不同的。举个例子，ul元素有缩进样式。在Firefox浏览器中，ul元素的缩进是通过padding实现的；而在IE浏览器中，ul元素的缩进是通过margin来实现的。再比如button元素，在IE、chrome、Firefox等浏览器中的默认样式也是不同的。图4-1为不同浏览器下的表单按钮。



图4-1　不同浏览器下的表单按钮

浏览器默认样式的不同，往往给我们的开发带来很大的麻烦，并且影响开发效率。为了解决这个问题，一个比较好的方法就是：去除元素在浏览器的默认样式（CSS reset）。

我们可以通过去除元素在浏览器的默认样式，使得HTML元素具有相同的初始样式，然后再对元素进行统一定义，就可以让页面在不同的浏览器中产生相同的显示效果。

4.5.3　如何使用CSS reset

说起去除浏览器默认样式，有些小伙伴可能会想到以下方法：

```
*{padding:0;margin:0;}
```

在实际开发中，并不建议使用这个方法。因为这个方法性能非常低，它把所有元素的padding和margin都去掉了，然而像表格元素（或者input元素）的margin和padding我们是不希望去掉的。此外，它只能消除默认的padding和margin，像ul的列表项符号、em的斜体、strong的加粗等却没有去除。

不过，在测试学习的过程中，我们可以使用“*{padding:0;margin:0;}”。如果在实际开发过程中，我们推荐CSS专家Eric Meyer的重置样式表，这是国内外流行的重置样式表。

以下是Eric Meyer发布的最新版的重置样式表（2011年1月26日发布），参考于http://meyerweb.com/eric/tools/css/reset/。

Eric Meyer的CSS reset完整代码如下：

```
/* http://meyerweb.com/eric/tools/css/reset/
   v2.0 | 20110126
   License: none (public domain)
*/
html, body, div, span, applet, object, iframe,
h1, h2, h3, h4, h5, h6, p, blockquote, pre,
a, abbr, acronym, address, big, cite, code,
del, dfn, em, img, ins, kbd, q, s, samp,
small, strike, strong, sub, sup, tt, var,
b, u, i, center,
dl, dt, dd, ol, ul, li,
fieldset, form, label, legend,
table, caption, tbody, tfoot, thead, tr, th, td,
article, aside, canvas, details, embed,
figure, figcaption, footer, header, hgroup,
menu, nav, output, ruby, section, summary,
time, mark, audio, video {
    margin: 0;
    padding: 0;
    border: 0;
    font-size: 100%;
    font: inherit;
    vertical-align: baseline;
}
/* HTML5 display-role reset for older browsers */
article, aside, details, figcaption, figure,
footer, header, hgroup, menu, nav, section {
 display: block;
}
body {
    line-height: 1;
}
ol, ul {
    list-style: none;
}
blockquote, q {
    quotes: none;
}
blockquote:before, blockquote:after,
q:before, q:after {
    content: ‘’;
    content: none;
}
table {
    border-collapse: collapse;
    border-spacing: 0;
}
```

上面这段代码是我们推荐的CSS reset代码，它可以去除常见HTML元素的默认样式。这个重置样式表包括了最新的HTML5元素，并且删除了过时的HTML元素。在实际开发中，我们建议大家使用CSS reset，这样我们就不会被元素的默认样式所干扰，从而随心所欲地定义自己的样式。

此外对于CSS reset，我们特别需要注意以下几点。

（1）CSS reset代码必须在其他CSS之前引入，道理很简单：浏览器对CSS代码是从上到下来解析的。只有把CSS reset放在前面，才有意义。

（2）Eric Meyer建议此CSS reset代码是应该根据个人需求不同来定义，例如有的页面不会用到address、code元素，直接把这两个元素剔除即可。因此，我们并不推荐大家直接把这段CSS reset代码简单地原样复制粘贴到自己的CSS中，而是根据自己的实际开发需求来定制，正所谓“滥用不如不用”。

（3）Eric Meyer建议此版本的CSS reset代码也有很多不足，例如div、li、code就没有padding和margin。因此，我们应该重新审视并修订这段代码再拿来用。

（4）目前优秀互联网企业的网站使用CSS reset越来越少，国外也有些优秀的网页设计师已经开始表达自己“不使用CSS reset”的观点。对于真正的实战开发来说，CSS reset也可以说是可有可无的。因此，对于是否使用CSS reset也应该根据实际开发需求来决定。

【疑问】

为什么浏览器要定义元素的默认样式呢？如果默认样式不存在，岂不是更好？这样既可以增强页面的兼容性了，又可以减少使用CSS reset来重置默认样式的繁琐。

其实浏览器之所以要存在默认样式，目的在于保证没有样式表的页面也能够正常显示。此外，HTML元素的默认样式往往跟它的语义挂钩。我们知道，一个页面在“CSS裸奔”下有很好的可读性，这也是元素默认样式起的作用。

第5章　盒子模型

5.1　CSS盒子模型

在“CSS盒子模型”理论中，页面中的所有元素都可以看成一个盒子，并且占据着一定的页面空间。

一个页面由很多这样的盒子组成，这些盒子之间会互相影响，因此掌握盒子模型需要从两个方面来理解：一是理解单独一个盒子的内部结构，二是理解多个盒子之间的相互关系。

每个元素都看成一个盒子，盒子模型是由content（内容）、padding（内边距）、margin（外边距）和border（边框）这四个属性组成的。此外，在盒子模型中，还有宽度width和高度height两大辅助性属性。

记住，是所有的元素都可以看成一个盒子！

图5-1所示为一个CSS盒子模型的内部结构。

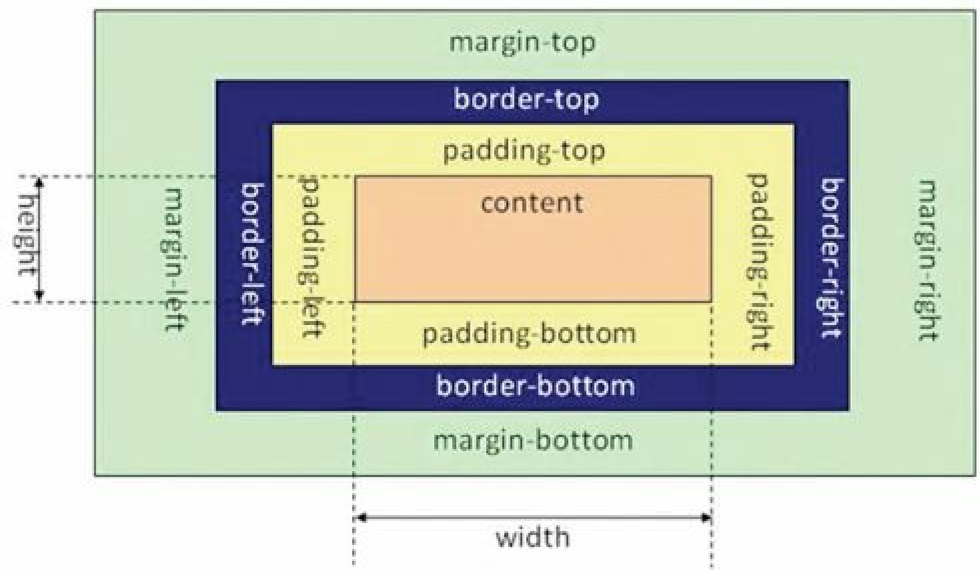


图5-1　CSS盒子模型

从上图中我们可以得出盒子模型的属性如表5-1所示。

表**5-1 CSS**盒子模型四个属性

属　　性	说　　明
border	（边框）元素边框
margin	（外边距）用于定义页面中元素与元素之间的距离
padding	（内边距）用于定义内容与边框之间的距离
content	（内容）可以是文字或图片

其中，padding称为“内边距”，也常常称为“补白”。图中的margin-top指的是顶部外边距、margin-right指的是右部外边距，以此类推。

1．内容区

内容区是CSS盒子模型的中心，它呈现了盒子的主要信息内容，这些内容可以是文本、图片等多种类型。内容区是盒子模型必备的组成部分，其他的三部分都是可选的。

内容区有三个属性：width、height和overflow。使用width和height属性可以指定盒子内容区的高度和宽度。在这里注意一点，width和height这两个属性是针对内容区而言，并不包括padding部分。

当内容信息太多而超出内容区所占范围时，可以使用overflow溢出属性来指定处理方法。

2．内边距

内边距，指的是内容区和边框之间的空间，可以看做是内容区的背景区域。

关于内边距的属性有五种，即padding-top、padding-bottom、padding-left、padding-right以及综合了以上四个方向的简洁内边距属性padding。使用这五种属性可以指定内容区域各方向边框之间的距离。

3．边框

在CSS盒子模型中，边框跟我们之前学过的边框是一样的。

边框属性有border-width、border-style、border-color以及综合了三类属性的快捷边框属性border。

其中border-width指定边框的宽度，border-style指定边框类型，border-color指定边框的颜色。

“border-width:1px;border-style:solid;border-color:gray;”等价于“border:1px solid gray;”。

4．外边距

外边距，指的是两个盒子之间的距离，它可能是子元素与父元素之间的距离，也可能是兄弟元素之间的距离。

外边距使得元素之间不必紧凑地连接在一起，是CSS布局的一个重要手段。

外边距的属性也有五种，即margin-top、margin-bottom、margin-left、margin-right以及综合了以上四个方向的简洁内边距属性margin。

同时，CSS允许给外边距属性指定负数值，当指定负外边距值时，整个盒子将向指定负值的相反方向移动，以此可以产生盒子的重叠效果。这就是传说中的“负margin技术”，我们将会绿叶学习网的CSS进阶教程中给读者详细讲解这一个高大上的技术喔。

内容区、内边距、边框、外边距这几个概念可能比较抽象，建议大家打好基础再来学习本章内容。对于基础知识可以参考本书的姊妹篇《Web前端开发精品课HTML和CSS基础教程》。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>CSS盒子模型</title>
    <style type="text/css">
        #main
        {
            display:inline-block;/*将块元素转换为inline-块元素*/
            border:1px solid #CCCCCC;
        }
        .lvye
        {
            display:inline-block; /*将块元素转换为inline-块元素*/
            padding:20px;
            margin:40px;
            border:1px solid red;
            background-color:#FCE9B8;
        }
        span{border:1px solid blue;background-color:#C5FCDF;}
    </style>
</head>
<body>
    <div id="main">
```

```
        <div class="lvye"><span>绿叶学习网</span></div>
    </div>
</body>
</html>
```

在浏览器预览效果如图5-2和图5-3所示。

分析：

我们把class为lvye的div层看做一个盒子，则浅蓝色部分为“内容区”，浅红色部分为“内边距区”，红色边框与灰色边框之间的空白为“外边距区”，红色的边框为该盒子的边框。



图5-2　CSS盒子模型实例



图5-3　CSS盒子模型实例分析

当然，我们也可以把最外层的id为main的div层看做一个盒子（因为所有HTML元素都可以看做一个盒子来理解），最外层的id为main的div层添加内边距和外边距，然后思考一下该盒子的“内容”“外边距”“内边距”和“边框”分别是什么？

我们从下面图5-4所示，可以很直观地理解CSS盒子模型，读者细细体会一下。

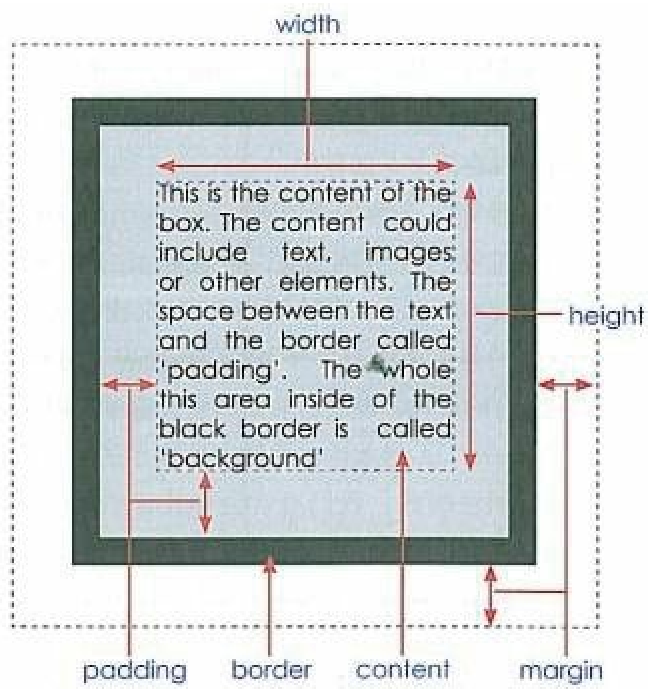


图5-4　CSS盒子模型分析图

在这里，给大家温习了一下CSS盒子模型的内容，对于border、padding、margin这几个属性的详细语法就不展开了，具体请查看本书的姊妹篇《Web前端开发精品课HTML和CSS基础教程》相关章节。在接下来的章节中，我们只注重讲解开发技巧以及一些深入的内容。

5.2　深入border

对于border属性，只有一个方面值得介绍的，那就是“border:0”和“border:none”的区别。

“border:0”与“border:none”的区别主要体现在两点：性能差异和兼容差异。

5.2.1　性能差异

1．border:0

“border:0”表示把border定义为0px。虽然0px在页面上看不见，但是浏览器依然会对border进行渲染，渲染之后，实际上是一个像素为“0”的border。

也就是说，“border:0”需要占用内存。

2．border:none

“border:none”表示把border定义为“none（无）”，浏览器解析“border:none”时并不会作出渲染。

也就是说，“border:none”不需要占用内存。

5.2.2　兼容差异

兼容性差异只存在于IE6和IE7的<input type=”button”/>标签以及<button>标签中，其他浏览器不存在兼容问题。

“border:0”在所有浏览器中的效果都一样，都是把边框隐藏（不是去掉），如图5-5所示。

“border:none”对IE6和IE7按钮的边框无效，在其他浏览器则会去掉按钮的边框，如图5-6所示。

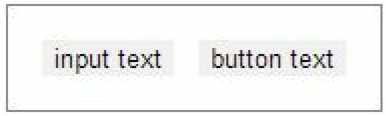


图5-5　“border:0”的按钮效果



图5-6　“border:none”在IE6/IE7中按钮的边框无效

由于IE6和IE7已经逐渐被抛弃了，因此我们不需要纠结“border:0”与“border:none”的兼容问题。

在实际开发中，对于“border:0”与“border:none”我们不需要纠结那么多，用哪个都差不多。兼容方面就不说了，经过测试，两者在性能方面对页面渲染速度的差别并不大。

5.3　深入padding

内边距padding，又常常称为“补白”，它指的是内容区到边框之间的那一部分，如图5-7所示。

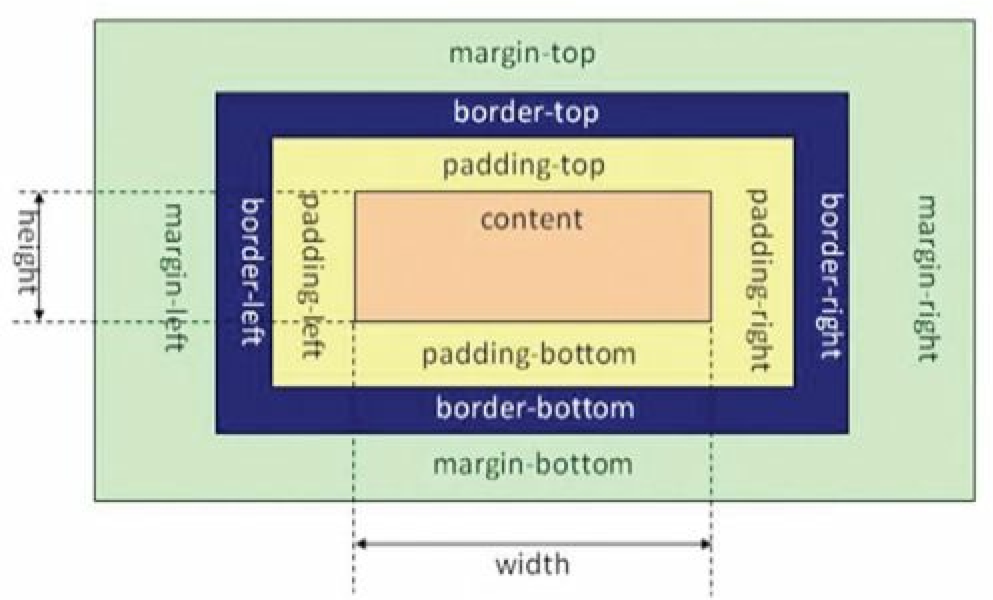


图5-7　CSS盒子模型

对于padding这个属性，没多少东西可以探讨。不过在实际开发中，关于背景图片的使用有时会涉及padding的一个小技巧。

像图5-8这种效果，我们都是使用“背景图片+padding”来实现的，如图5-9所示，分析如下：

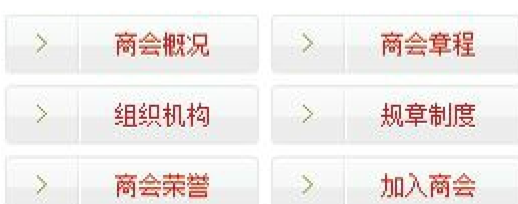


图5-8　背景图片与padding案例

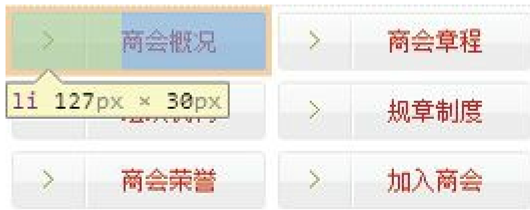


图5-9　背景图片与padding案例分析

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        div
        {
            background-image:url("images/bg.gif");
            width:127px;
            height:30px;
            line-height:30px;
            font-size:12px;
            color:Red;
        }
    </style>
</head>
<body>
    <div>商会状况</div>
</body>
</html>
```

在浏览器预览效果如图5-10所示。



图5-10　padding技巧实例

分析：

在这里我们可以看到，当一个元素使用背景图的时候，该元素的文字内容会停留在左边，如果想要达到预期效果，可以使用padding来实现，修改后的CSS代码如下：

```
div
{
    background-image:url("images/bg.gif");
    width:72px;    /*127-55=72*/
    height:30px;
line-height:30px;
    padding-left:55px;
    font-size:12px;
    color:Red;
}
```

5.4　外边距叠加

有过开发经历的小伙伴们可能会碰过这种情况：有相邻的两个块元素A和B，上面的为A，下面的为B。其中A定义了一个margin-bottom，B定义了一个margin-top。在浏览器预览效果中，我们会发现A和B之间的垂直距离明显小于margin-bottom和margin-top相加之和。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        div
        {
            height:60px;
            line-height:60px;
            text-align:center;
            font-size:30px;
            color:White;
            background-color:Purple;
        }
        #first{margin-bottom:20px;}
        #second{margin-top:30px;}
    </style>
</head>
<body>
    <div id="first">A</div>
    <div id="second">B</div>
</body>
</html>
```

在浏览器预览效果如图5-11所示。

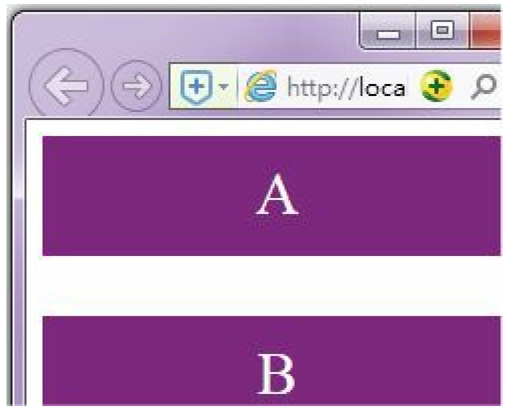


图5-11　A和B之间的垂直距离小于两者之和

分析：

在这里，A的margin-bottom为20px，B的margin-top为30px，但是A和B之间的间距并不是50px。不知道原因所在的小伙伴们还以为浏览器有bug。其实，这个现象是由外边距叠加所引起的。

外边距叠加，又称“margin叠加”，指的是当两个垂直外边距相遇时，这两个外边距将会合并为一个外边距，即“二变一”。其中，叠加之后的外边距高度等于发生叠加之前的两个外边距中的最大值。

对于外边距叠加，我们分为三种情况来讨论：同级元素、父子元素和空元素。

5.4.1　外边距叠加的三种情况

1．同级元素

当一个元素出现在另外一个元素上面的时候，第一个元素的下边距（margin-bottom）将会与第二个元素的上边距（margin-top）会发生合并，如图5-12所示。

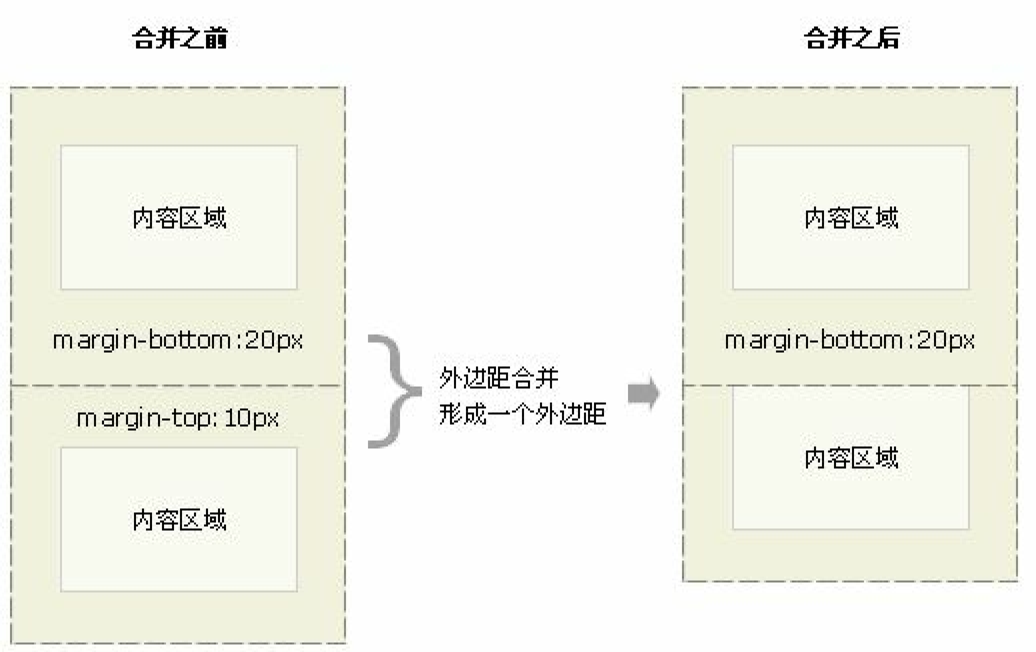


图5-12　同级元素的外边距叠加

2．父子元素

当一个元素包含在另外一个元素中时（父子关系），假如没有内边距padding或边框border把外边距分隔开的话，父元素和子元素的相邻上下外边距也会发生合并，如图5-13所示。

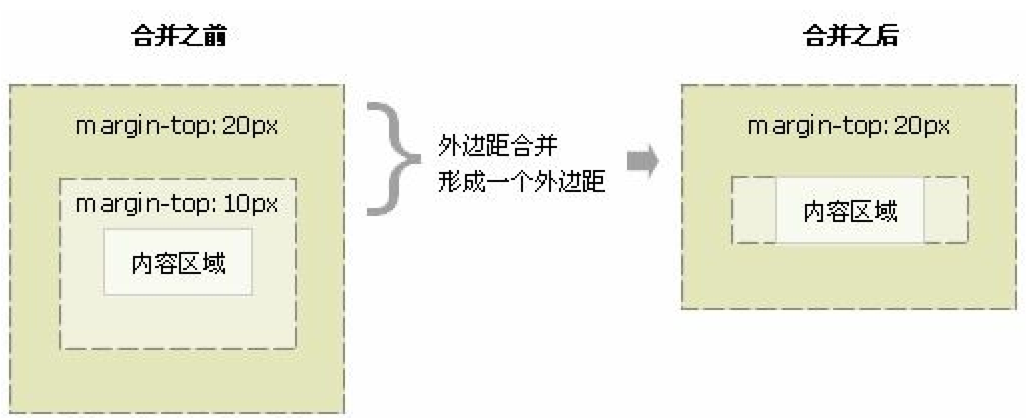


图5-13　父子元素的外边距叠加

3．空元素

当一个空元素有上下外边距时，如果没有border或者padding，则元素的上外边距与下外边距会发生合并。

空元素，指的是没有子元素或者没有文字内容的元素，例如
、<hr/>等。

如果空元素的外边距碰到另外一个元素的外边距，它们也会发生合并。

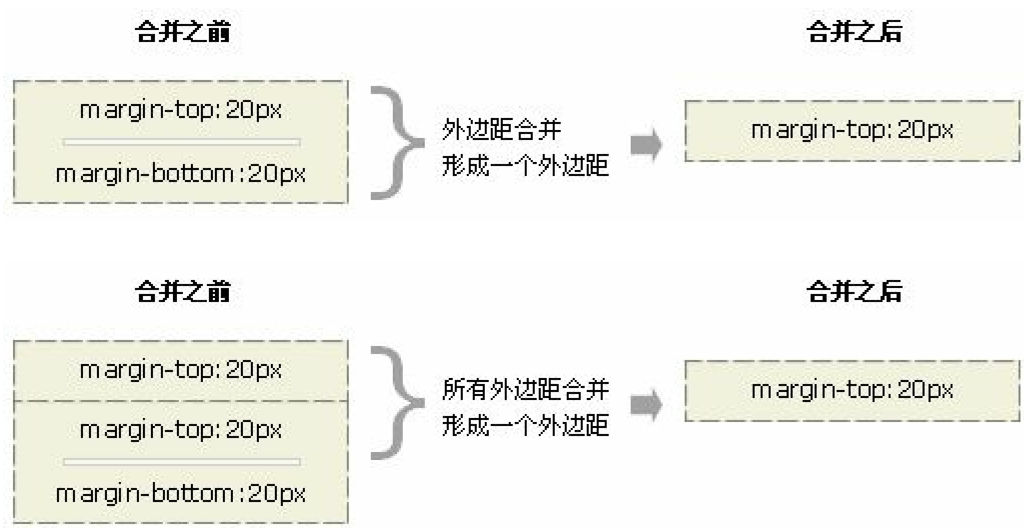


图5-14　空元素的外边距叠加

外边距叠加只有三种情况：同级元素、父子元素和空元素。此外对于外边距叠加，我们还需要注意以下几点：

（1）水平外边距永远不会有叠加，水平外边距指的是margin-left和margin-right。

（2）垂直外边距只会在以上三种情况下会叠加，垂直外边距指的是margin-top和margin-bottom。

（3）外边距叠加之后的外边距高度等于发生叠加之前的两个外边距中的最大值。

（4）外边距叠加针对的是block以及inline-块元素，不包括inline元素。因为inline元素的margin-top和margin-bottom设置无效。

5.4.2　外边距叠加的意义

下面是一个文本型页面，第一个段落上面的空间等于段落的margin-top。如果没有外边距叠加，第一个段落之后的所有段落之间的外边距都是相邻的margin-top和margin-bottom之和，这样就跟第一个段落显得不一致了。

如果发生外边距叠加，段落之间的margin-top和margin-bottom就会合并在一起，这样就跟第一个段落显得一致了。

CSS定义外边距叠加的初衷就是为了更好地对文章进行排版，如图5-15所示。了解这一点，对于我们理解和记忆外边距叠加很有帮助。

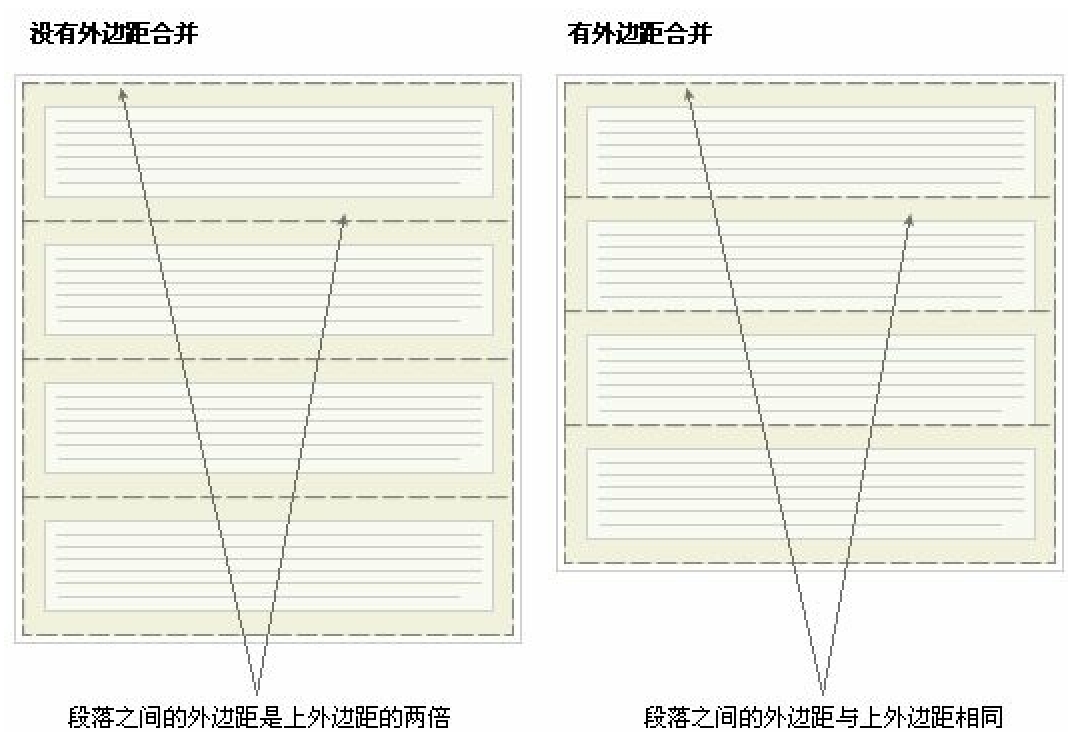


图5-15　外边距叠加的意义

当然，了解外边距叠加的原理，能给我们解决不少疑惑。此外在实际开发中，给大家一个建议：最好统一使用margin-top或margin-bottom，不要混合使用，从而降低出现问题的风险。当然这在技术上并不是必需的，但却是一个良好的习惯。

外边距叠加的原理比较复杂，跟BFC（块级格式上下文）有着密切

的关系。这一节我们只是简单介绍了最基本的东西，对于BFC我们在后续章节会详细介绍。

5.5　负margin技术

在CSS中，margin属性取值可以为正数，也可以为负数。margin无论取值为正数还是负数，都可以让当前元素或者周围元素进行移动。但是margin为正数和margin为负数，这两者却有着很大的不同。对于margin为正数的情况，我们不在此展开，因为这种情况我们已经接触得很多了。这一节给大家详细介绍一下margin为负数（也就是负margin）的相关技巧。

5.5.1　负margin简介

当margin为负数的时候，对普通文档流元素和对浮动元素的影响是不一样的，如图5-16所示。负margin对普通文档流元素的影响，我们分两种情况。

（1）当元素的margin-top或者margin-left为负数时，“当前元素”会被拉向指定方向。

（2）当元素的margin-bottom或者margin-right为负数时，“后续元素”会被拉向指定方向。

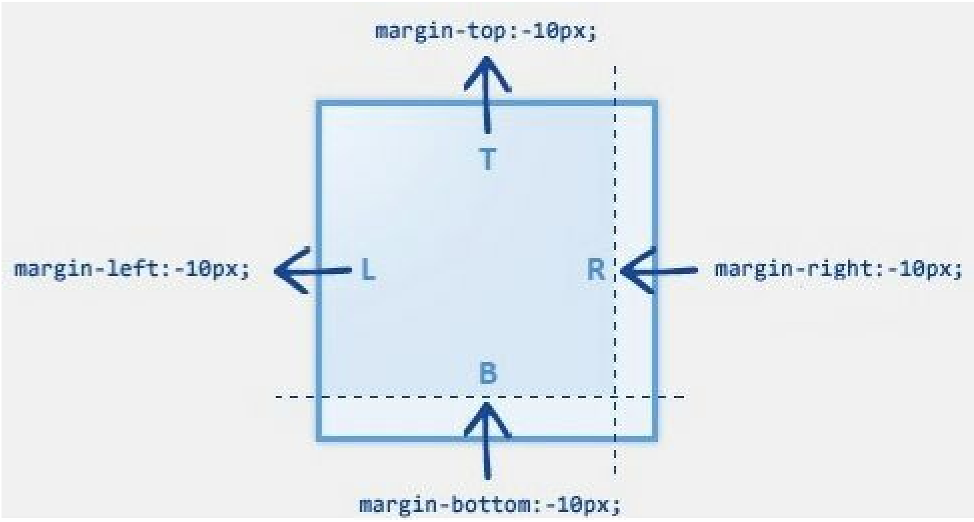


图5-16　不同方向的负margin分析图

从图5-16中我们可以看出，margin-top和margin-left将“当前元素”拉出，然后覆盖“其他元素”。margin-bottom和margin-right是将“后续元素”拉进，然后覆盖“当前元素”。

负margin对浮动元素的影响，跟负margin对普通文档流元素的影响其实是差不多的。唯一不太一样的是，浮动元素可以向左也可以向右。因此对于浮动元素，我们只需要比普通文档流元素多关注一点，那就是浮动元素的“流动的方向”。

其实，对于负margin引起的元素移动方向我们很容易忘记。不过也不用担心，在实际开发过程中，我们稍微测试一下就很容易知道了。

举例：margin-top或margin-bottom为负数

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        #wrapper div
        {
            width:300px;
            height:60px;
            line-height:60px;
            font-size:21px;
            font-weight:bold;
            text-align:center;
            color:White;
        }
        #first{background-color:Red;}
        #second{background-color:Purple;}
        #third{background-color:Blue;}
    </style>
</head>
<body>
    <div id="wrapper">
        <div id="first">1</div>
        <div id="second">2</div>
        <div id="third">3</div>
    </div>
```

```
</body>
</html>
```

在浏览器预览效果如图5-17所示。



图5-17　未设置　margin-top或margin-bottom

分析：

当我们为第二个div添加“margin-top:-30px;”时，在浏览器预览效果如图5-18所示，我们可以看到“当前元素”（第二个div）被拉向上方。

当我们为第二个div添加“margin-bottom:-30px;”时，在浏览器预览效果如图5-19所示，我们可以看到“后续元素”（第三个div）被拉向上方。

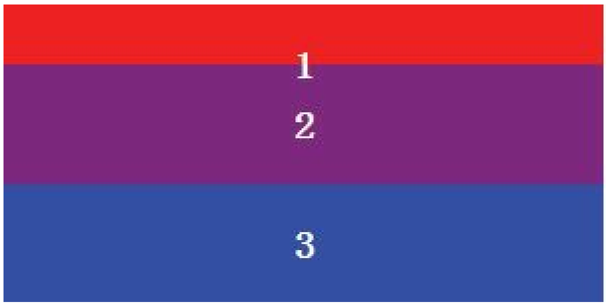


图5-18　第二个div设置margin-top为-30px

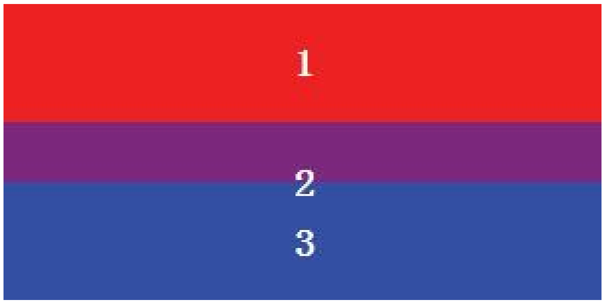


图5-19　第二个div设置margin-bottom为-20px

举例：margin-left或margin-right为负数

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        /*去除inline-块元素之间的间距*/
        #wrapper{font-size:0;}
        #wrapper div
        {
            display:inline-block;
            width:60px;
            height:60px;
            line-height:60px;
            font-size:21px;
            font-weight:bold;
            text-align:center;
            color:White;
        }
        #first{background-color:Red;}
        #second{background-color:Purple;}
        #third{background-color:Blue;}
    </style>
</head>
<body>
    <div id="wrapper">
        <div id="first">1</div>
        <div id="second">2</div>
        <div id="third">3</div>
    </div>
</body>
```

```
</html>
```

在浏览器预览效果如图5-20所示。



图5-20　未设置margin-left或margin-right

分析：

当我们为第二个div添加“margin-left:-30px;”时，在浏览器预览效果如图5-21所示，我们可以看到“当前元素”（第二个div）被拉向左方。

当我们为第二个div添加“margin-right:-30px;”时，在浏览器预览效果如图5-22所示，我们可以看到“后续元素”（第三个div）被拉向左方。



图5-21　第二个div设置margin-left为-30px

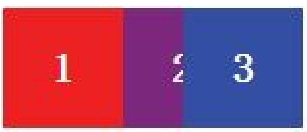


图5-22　第二个div设置margin-right为-30px

5.5.2　负margin技巧

负margin的使用很灵活，常用技巧有四个。

（1）图片与文字对齐。

（2）自适应两列布局。

（3）元素垂直居中。

（4）tab选项卡。

1. 图片与文字对齐

当图片与文字排在一起的时候，在底部水平方向上往往都是不对齐的，这是因为图片与文字默认是基线对齐（vertical-align:baseline）。如果想要实现图片与文字底部水平方向对齐，除了使用“vertical-align:text-bottom”这个方法之外，还可以使用兼容性比较好的负margin来实现。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
</head>
<body>
    <img src="images/baidu.png" alt=""/>百度LOGO
</body>
</html>
```

在浏览器预览效果如图5-23所示。



图5-23　图片与文字在一起的效果

分析：

从图5-23中我们可以看出，默认情况下图片与文字在底部水平方向上是不对齐的。我们在CSS中添加“img{margin:0 3px -3px 0;}”之后，在

浏览器预览效果如图5-24所示。实际上，“margin{0 3px -3px 0}”可以看成一条公式般的东西，大家记住就行。



图5-24　图片与文字对齐效果

2. 自适应两列布局

自适应两列布局，指的是在左右两列中，其中有一列的宽度为自适应，另外一列宽度是固定的。如果使用float，一般只能实现固定的左右两列布局，并不能实现其中一列为自适应的布局。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        #content,#sidebar
        {
            float:left;
            color:white;
        }
        #content
        {
            width:100%;
            margin-right:-200px;
            background-color:Red;
        }
        #sidebar
        {
            width:200px;
            background-color:Purple;
        }
        /* 防止浏览器可视区域宽度不足时发生文本重叠*/
        #content p {margin-right:210px;}
        /* 它是 200px + 10px, 10px是他们的间距*/
```

```
    </style>
</head>
<body>
    <div id="content"> <p>这是主体部分，自适应宽度</p> </div>
    <div id="sidebar"> <p>这是侧边栏部分，固定宽度 </p> </div>
</body>
</html>
```

在浏览器预览效果如图5-25所示。

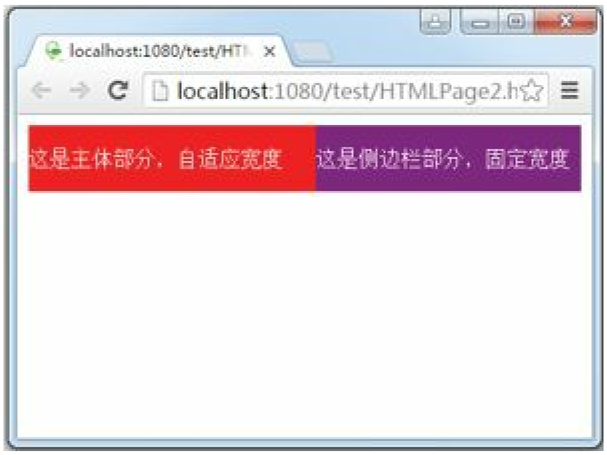


图5-25　负margin实现自适应两列布局

分析：

我们改变浏览器的宽度，就可以很容易地看出自适应两列布局实际效果。WordPress经典的两栏自适应布局就是使用这种方法来实现的。

3. 元素垂直居中

想要实现块元素的垂直居中一般来说比较麻烦，不过有一个经典的方法，就是使用position结合负margin来实现。具体做法是：首先给父元素写上“position:relative”，这样做是为了给子元素添加“position:absolute”的时候不会被定位到“外太空”去。然后给子元素添加如下属性：

```
position:absolute;
```

```
top:50%;
left:50%;
```

之后再添加如下属性：

```
margin-top:"height值一半的负值";
margin-left:"width值一半的负值"。
```

语法：

```
父元素
{
    position:relative;
}
子元素
{
    position:absolute;
    top:50%;
    left:50%;
    margin-top:"height值一半的负值";
    margin-left:"width值一半的负值";
}
```

说明：

position这种方法是万能的，也就是不仅可以用于块元素（block），还可以用于inline元素和inline-块元素。对于margin-top和margin-left为什么要这样定义，大家自行画个草稿图就能理解。

举例：

```
<!DOCTYPE html>
```

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        #father
        {
            position:relative;
            width:200px;
            height:160px;
            border:1px dashed gray;
        }
        #son
        {
            position:absolute;
            top:50%;
            left:50%;
            margin-top:-30px;
            margin-left:-50px;
            width:100px;
            height:60px;
            background-color:Red;
        }
    </style>
</head>
<body>
    <div id="father">
        <div id="son"></div>
    </div>
</body>
</html>
```

在浏览器预览效果如图5-26所示。

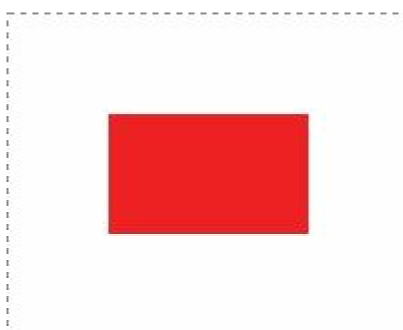


图5-26　负margin实现元素垂直居中

分析：

更多关于元素的垂直居中技巧，我们在后面的“13.2　垂直居中”一节会详细介绍。

4．tab选项卡

tab选项卡效果是一种十分节省页面空间的方式，在实际开发中经常会用到。像图5-27中的选项卡，关键是使用“margin-top:-1px”来解决选项卡下边框显示问题。由于tab选项卡涉及JavaScript内容比较多，因此不在此展开。有兴趣的同学可以使用调试工具（如Firebug）查看一下绿叶学习官网首页的tab选项卡，可以学到很多东西。

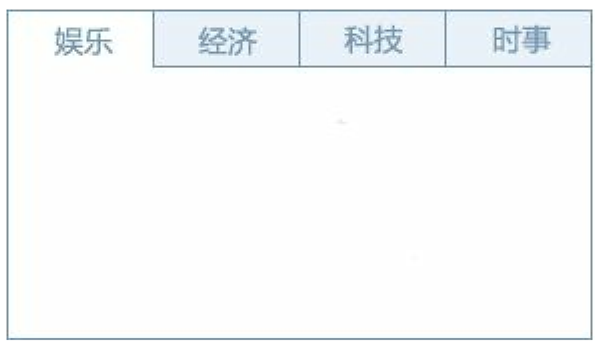


图5-27　负margin实现tab选项卡

5.6　overflow

W3C标准指出，通常一个盒子的内容是被限制在盒子边框之内的。但是有时也会产生溢出，也就是部分或者全部内容跑到盒子边框之外。

在CSS中，我们可以使用overflow属性来定义当内容溢出元素边框时发生的事情。

语法：

```
overflow:属性值;
```

说明：

overflow属性取值如表5-2所示。

表**5-2　overflow**属性取值

属　性　值	说　　明
visible	默认值，若内容溢出，则溢出内容可见
hidden	若内容溢出，则溢出内容隐藏
scroll	若内容溢出，则显示滚动条
auto	auto跟scroll很相似，不同的是auto值在盒子需要的时候给它一个滚动条

对于overflow属性，最常见的三个用途如下。

（1）使用“overflow:scroll”显示滚动条。

（2）使用“overflow:hidden”来隐藏内容，以免影响布局。

（3）使用“overflow:hidden”来清除浮动。

“overflow:hidden”会使得超出元素的部分自动隐藏，这样处理不好的一点就是这部分的内容显示不完全，比如图片只显示了一部分。不过“overflow:hidden”使得内容超出时不会影响页面整体布局，这是它的好处。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        #content
        {
            width:200px;
            height:160px;
            border:1px solid gray;
        }
    </style>
</head>
<body>
    <div id="content">水陆草木之花，可爱者甚蕃。晋陶渊明独爱菊。自李唐来，世人甚
</body>
</html>
```

在浏览器预览效果如图5-28所示。

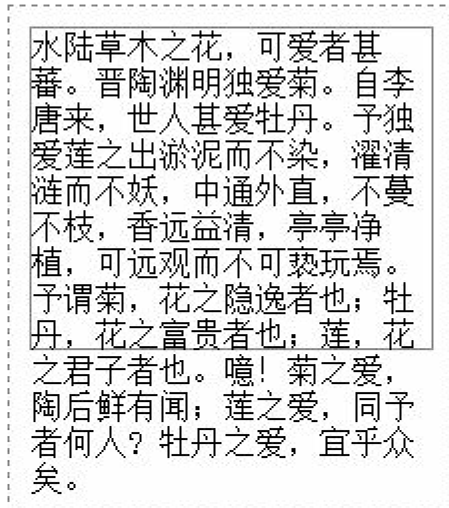


图5-28　overflow为默认值的效果

分析：

默认情况下，div的overflow属性值为visible，当我们在CSS中为div添加“overflow:hidden”后，在浏览器预览效果如图5-29所示。

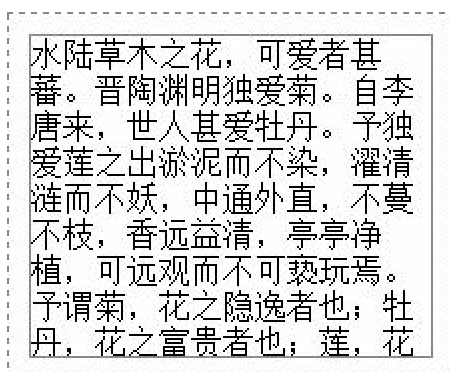


图5-29　overflow为hidden的效果

当我们在CSS中为div添加“overflow:scroll”后，在浏览器预览效果如图5-30所示。

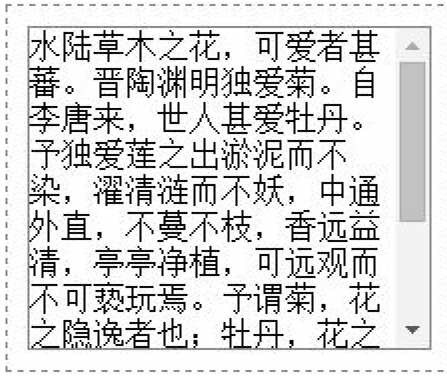


图5-30　overflow为scroll的效果

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        #wrapper
        {
            width:200px;
            border: 1px solid black;
        }
        #first,#second
        {
            width:80px;
            height:40px;
            border:1px solid red;
        }
        #first{float:left;}
        #second{float:right;}
    </style>
</head>
<body>
    <div id="wrapper">
        <div id="first"></div>
        <div id="second"></div>
    </div>
</body>
</html>
```

在浏览器预览效果如图5-31所示。

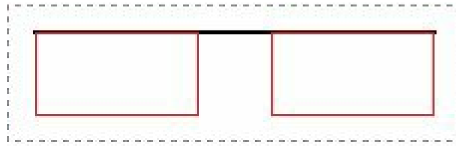


图5-31　浮动引起的父元素高度塌陷

分析：

当我们为id为wrapper的div元素添加“overflow:hidden”之后，在浏览器预览效果如图5-32所示。从中可以看出，浮动已经被清除了。

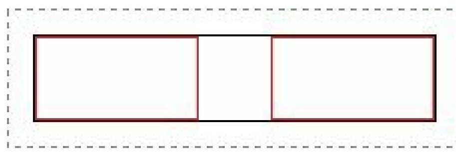


图5-32　overflow:hidden清除浮动

使用clear属性来清除浮动的缺点是增加一个多余的标签，而使用“overflow:hidden”清除浮动则不需要。不过“overflow:hidden”是一个小炸弹，它会将超出父元素部分的内容隐藏，有时候这并不是我们预期想要的效果。

第6章　display属性

6.1　块元素和行内元素

在接触display属性之前，我们先来回顾一下块元素和行内元素的基本概念。

在HTML的学习中我们可能发现，在浏览器的显示效果中，有些元素是独占一行的，别的元素不能跟这个元素位于同一行，如h1～h6、p、div等；而有些元素可以跟其他元素位于同一行，如strong、em、u等。

HTML元素根据表现形式，常见的可以分为两类。

（1）块元素（block）。

（2）行内元素（inline）。

block和inline是最常见的两种形式，当然HTML元素类型除了这两种，还有inline-block、table-cell等。这一节我们先重点学习block和inline这两个元素类型。

6.1.1　块元素

块元素在浏览器默认显示状态下将占据整行，排斥其他元素与其位于同一行。块元素一般为矩形，可以容纳行元素和其他的块元素。常见的块元素见表6-1。

表**6-1** 常见块元素

块　元　素	说　　明

div	div层
h1～h6	1到6级标题
p	段落，会自动在其前后创建一些空白
hr	分割线
ol	有序列表
ul	无序列表

表中只是列出了HTML入门常见的块元素，并不是全部。

块元素具有如下特点。

（1）独占一行，排斥其他元素跟其位于同一行，包括块元素和行内元素。

（2）块元素内部可以容纳其他块元素或行元素。

（3）可以定义高度（height），也可以定义宽度（width）。

（4）可以定义四个方向的margin属性。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>块元素和行内元素</title>
```

```
</head>
<body>
    <div>
        <h3>绿叶学习网</h3>
        <p>让这里的一切成为衬托你成功的绿叶</p>
        <strong>绿叶学习网</strong><em>让这里的一切成为衬托你成功的绿叶</em>
    </div>
</body>
</html>
```

在浏览器预览效果如图6-1所示。

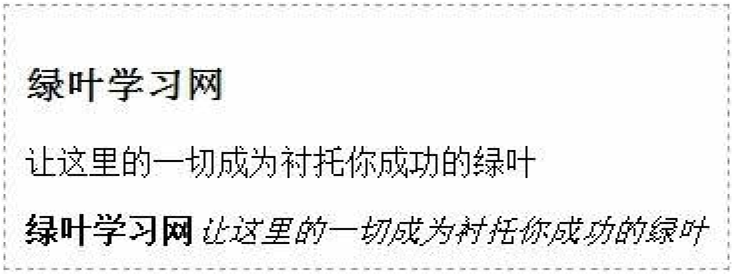


图6-1　块元素和行内元素

分析：

我们为每一个元素加入虚线框来分析它们的结构，如图6-2所示。



图6-2　块元素和行内元素分析图

大家很容易看到以下特点。

（1）h3和p是块元素，它们都是独占一行的，并且排斥任何元素（包括块元素和行内元素）跟它们位于同一行；而strong和em是行内元素，相邻两个行内元素是可以位于同一行的。

（2）h3、p、strong和em这几个元素是位于div元素内部的，也就是块元素内部可以容纳其他块元素或行元素。

6.1.2　行内元素

行内元素与块元素恰恰相反。行内元素默认显示状态可以与其他行内元素共存在同一行。常见的行内元素见表6-2。

表**6-2** 常见行内元素

行 内 元 素	说　　明
strong	加粗强调
em	斜体强调
s	删除线
u	下划线
a	超链接
span	常用行级，可定义文档中的行内元素

行内元素具有如下特点。

（1）可以与其他行内元素位于同一行。

（2）行内内部可以容纳其他行内元素，但不可以容纳块元素，不然会出现无法预知的效果。

（3）无法定义高度（height），也无法定义宽度（width）。

（4）可以定义margin-left和margin-right，无法定义margin-top和margin-bottom。

对于行内元素的理解，我们重新回去查看块元素中的实例。

6.2　display简介

从上一节我们知道，除了block和inline，HTML元素还有inline-block、table、table-cell等类型。如果我们想要将元素从一个类型转换为另外一个类型，怎么办呢？

在CSS中，我们可以使用display属性来改变元素的类型。

语法：

```
display:属性值;
```

说明：

display属性取值如表6-3所示。

表**6-3**　**display**属性取值

属　性　值	说　　明
inline	行内元素
block	块元素
inline-block	行内块元素
table	以表格形式显示，类似于table元素

table-row	以表格行形式显示，类似于tr元素
table-cell	以表格单元格形式显示，类似于td元素
none	隐藏元素

除了上表这些，display还有list-item、run-in、compact等属性值。不过其他属性值我们极少用到，不需要去研究。我们只需要认真掌握上表中这几个，就可以走得很远了。

6.2.1　块元素

块元素，指的是块元素。块元素一般具有以下几个特点。

（1）独占一行，排斥其他元素跟其位于同一行，包括块元素和行内元素。

（2）块元素内部可以容纳其他块元素或行元素。

（3）可以定义宽度（width），也可以定义高度（height）。

（4）可以定义四个方向的margin。

6.2.2　inline元素

inline元素，指的是行内元素。行内元素一般具有以下几个特点。

（1）可以与其他行内元素位于同一行。

（2）行内内部可以容纳其他行内元素，但不可以容纳块元素，不然会出现无法预知的效果。

（3）无法定义高度（height），也无法定义宽度（width）。

（4）可以定义margin-left和margin-right，无法定义margin-top和margin-bottom。

6.2.3 inline-block元素

在CSS中，我们可以使用“display:inline-block”来将元素转换为行内块元素。行内块元素具有以下两个特点。

（1）可以定义width和height。

（2）可以与其他行内元素位于同一行。

也就是说inline-block元素既具备块元素的特点，也具备行内元素的特点。在HTML中，常见的inline-block元素有两个：img元素和input元素。对于这两个inline-block元素，我们一定要记住。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        span
        {
            display:inline-block;
            width:60px;
            height:100px;
            background-color:red;
        }
    </style>
</head>
<body>
    <span></span>
    <span></span>
    <span></span>
</body>
</html>
```

在浏览器预览效果如图6-3所示。

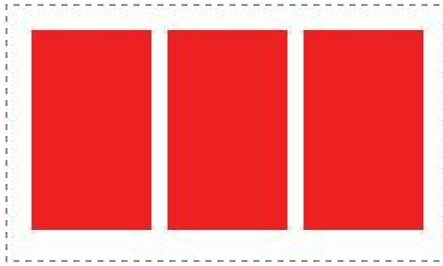


图6-3　display:inline-block实例

分析：

在实际开发中，我们可能经常需要为span等行内元素定义一定的width和height，此时应该考虑到“display:inline-block;”。此外，细心的小伙伴会发现，怎么inline-block元素之间还有间距呢？对于这个问题，我们在后面“去除inline-block元素间距”一节会给大家详细介绍。

有些新手很容易忘记inline-block类型的特点。我们都知道img就是inline-block类型元素，它可以定义width和height，还可以与其他行内元素（如span）位于同一行。如果我们不记得inline-block类型是怎样的，想一下img元素就知道了。

此外对于块元素，IE6和IE7不能识别“display:inline-block”，加不加“display:inline-block”对它们完全没有任何影响。解决方法是：在IE6和IE7中用“*display:inline;*zoom:1;”来代替“display:inline-block;”。对于行内元素，比如a、span等，display:inline-block不存在兼容问题，所有浏览器都可以识别，可以正常使用。当然IE6和IE7也逐渐淡出历史舞台了，这些了解一下即可。

这一节，我们把display属性常见的三个属性block、inline以及inline-block放在一起介绍，方便对比理解记忆。对于其他属性，我们接下来一一介绍。

6.3　display:none

6.3.1　display:none简介

在CSS中，我们可以使用“display:none”来隐藏元素。“display:none”用得很多，如二级导航、tab选项卡等地方都用到。不过一般情况下，“display:none”都是配合JavaScript来动态隐藏元素的。

对于“display:none”，我们需要注意以下两点。

（1）“display:none”一般用于JavaScript动态隐藏元素，被隐藏的元素不占据原来的位置空间。

（2）“display:none”不推荐用来隐藏一些对SEO关键的部分。因为对于搜索引擎来说，它会直接忽略“display:none”隐藏的内容，不把“display:none”隐藏的内容加入权重考虑。

对于第二点，我们在“7.2　深入text-indent”一节会给大家详细介绍。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        div
        {
            display:inline-block;
            width:60px;
            height:60px;
            line-height:60px;
            border:1px solid red;
            font-size:36px;
```

```
            text-align:center;
        }
    </style>
</head>
<body>
    <div id="first">A</div>
    <div id="second">B</div>
    <div id="third">C</div>
</body>
</html>
```

在浏览器预览效果如图6-4所示。

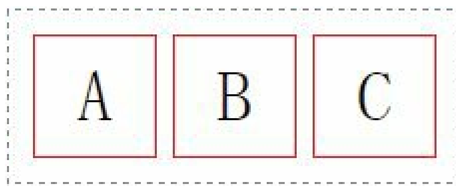


图6-4　第2个div未设置“display:none”

分析：

当我们给第2个div添加“display:none;”属性后，在浏览器预览效果如图6-5所示。

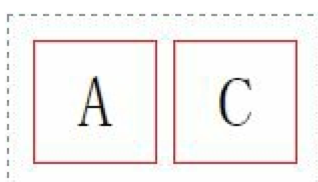


图6-5　第2个div设置“display:none”

我们会发现第二个div元素隐藏了，并且被隐藏的元素不再占据原来位置的空间。

6.3.2 “display:none”和“visibility:hidden”的区别

在CSS中，如果想要隐藏某一个元素，我们可以使

用“display:none”或者“visibility:hidden”来实现。但是这两者也有本质上的区别。

（1）“display:none”的元素被隐藏之后，不占据原来的位置。也就是说彻底地消失了，看不见也摸不着。

（2）“visibility:hidden”的元素被隐藏之后，依然占据原来的位置。也就是说并没有彻底消失，看不见但摸得着。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        #father
        {
            display:inline-block;
            border:1px solid silver;
            padding:10px;
        }
        #first,#second
        {
            display:inline-block;
            width:60px;
            height:60px;
            line-height:60px;
            font-size:36px;
            text-align:center;
            color:White;
            background-color:Red;
        }
    </style>
</head>
<body>
    <div id="father">
        <div id="first">A</div>
        <div id="second">B</div>
    </div>
</body>
```

```
</html>
```

在浏览器预览效果如图6-6所示。

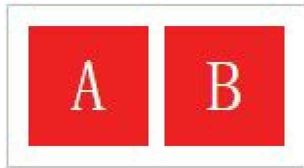


图6-6　A未设置“display:none”和“visibility:hidden”

分析：

当我们为id为first的div元素添加“display:none”后，在浏览器预览效果如图6-7所示。

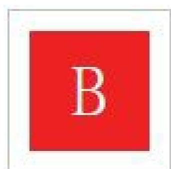


图6-7　A设置“display:none”

当我们为id为first的div元素添加“visibility:hidden”后，在浏览器预览效果如图6-8所示。

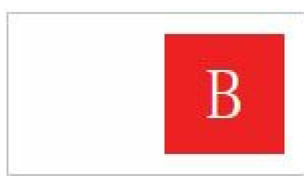


图6-8　A设置“visibility:hidden”

从上面我们可以看出，“display:none”或者“visibility:hidden”都可以隐藏元素，不过使用“display:none”的元素被隐藏之后不会占据原来的位置，而使用“visibility:hidden”的元素被隐藏之后会占据原来的位置。

6.4 display:table-cell

在CSS中，“display:table-cell”可以让元素以表格单元格的形式呈现。也就是说，table-cell类型的元素具备td元素的特点。

目前IE8+以及其他现代浏览器都支持此属性，不过在IE6/IE7中并不支持。考虑到IE6/IE7使用率越来越低的情况，我们还是果断使用“display:table-cell”这一布局神器。

“display:table-cell”非常强大，可以实现以下三种功能。

（1）图片垂直居中于元素。

（2）等高布局。

（3）自动平均划分元素，并且在一行显示。

6.4.1 图片垂直居中于元素

在CSS中，我们可以使用“display:table-cell”和“vertical-align:center”来实现大小不固定的图片的垂直居中效果。

语法：

```
父元素
{
    display:table-cell;
    vertical-align:center;
}
子元素{vertical-align:center;}
```

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        div
        {
            display:table-cell;
            width:150px;
            height:150px;
            border:1px solid gray;
            vertical-align:middle;
            text-align:center;
        }
        img{vertical-align:middle;}
        div+div{border-left:none;}
    </style>
</head>
<body>
    <div><img src="images/haizei1.png" alt=""/></div>
    <div><img src="images/haizei2.png" alt=""/></div>
    <div><img src="images/haizei3.png" alt=""/></div>
</body>
</html>
```

在浏览器预览效果如图6-9所示。



图6-9　“display:table-cell”实现图片垂直居中于元素

分析：

图片的水平居中可以使用“text-align:center”来实现，而图片的垂直

居中效果可以使用“display:table-cell”和“vertical-align:center”配合来实现。图片的水平居中和垂直居中我们在实际开发中也经常使用，例如绿叶学习网下面图片列表效果就是使用以上方法来实现的如图6-10所示。

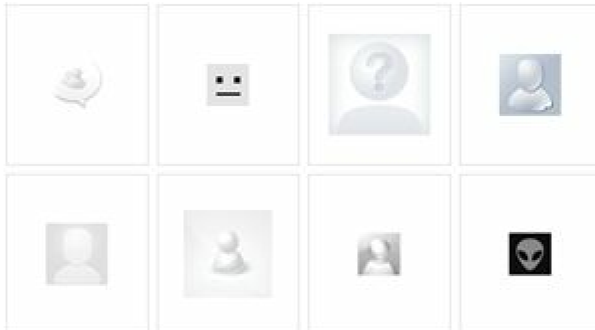


图6-10　绿叶学习网图片列表效果

6.4.2　等高布局

我们知道，同一行的单元格td元素高度是相等的。因此，table-cell元素也具备这个特点。根据这个特点，我们可以实现等高布局效果。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        #wrapper{display:table-row;}
        #img-box
        {
            display:table-cell;
            vertical-align:middle;     /*垂直居中*/
            text-align:center;         /*水平居中*/
            width:150px;
            border:1px solid red;
        }
        #text-box
        {
            display:table-cell;
```

```
            width:200px;
            border:1px solid red;
            border-left:none;
            padding:10px;
        }
    </style>
</head>
<body>
    <div id="wrapper">
        <div id="img-box">
            <img src="images/haizei1.png" alt=""/>
        </div>
        <div id="text-box">
              <span>《ONE PIECE》（海贼王、航海王）简称"OP"，是日本漫画家尾田荣一
        </div>
    </div>
</body>
</html>
```

在浏览器预览效果如图6-11所示。



图6-11 “display:table-cell”实现等高布局

分析：

在这个例子中，左右两个盒子我们都没有加上高度，而是由盒子内容撑开。但是我们会发现一个很有趣的现象：左右两个盒子高度相等，并且高度由两者高度最大值决定。这就是自适应的等高布局。

小伙伴们可能会问，这种自适应的等高布局有什么用呢？像下面好友动态页中的两栏布局中就可以用到这种自适应等高布局如图6-12所示。



图6-12　好友列表中的等高布局

上面这种布局是两栏的，左栏只有一个头像，右栏是内容。有些同学可能会为左右两栏定义相同的高度来实现。但是我们都知道好友动态右栏的内容多少往往都是不确定的，如果定义固定高度的话，内容超出了高度怎么办？这个时候使用上面介绍的等高的自适应布局就可以完美解决了。在自适应的等高布局中，左右两栏都不定义高度，而是由内容撑起来，左右两栏的高度都相同。这个技巧非常实用，建议大家一定要掌握。

6.4.3　自动平均划分元素

如果我们想要使用ul来实现下面这种布局，一般都会用float来实现，并且还得精准计算每一个li的宽度。但是如果给每个li元素都定义“display:table-cell”，我们就不用显得如此麻烦。而且会自动平均划分元素，并在一行显示如图6-13所示。

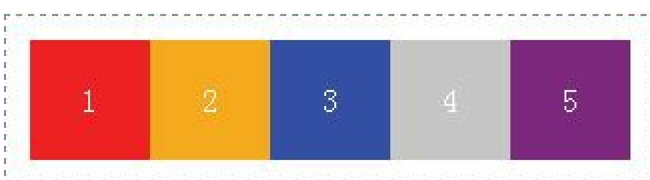


图6-13　宽度相等的元素

语法：

```
父元素{display:table};
子元素{display:table-cell;}
```

说明：

当父元素定义“display:table”而子元素定义“display:table-cell”时，如果给父元素一定的宽度，父元素宽度就会根据子元素的个数进行自动平均划分。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        *{padding:0;margin:0;}
        ul
        {
            list-style-type:none;
            display:table;
            width:300px;
        }
        li
        {
            display: table-cell;
            height:60px;
            line-height:60px;
            text-align:center;
            color:White;
        }
        ul li:nth-child(1){background-color:Red;}
        ul li:nth-child(2){background-color:Orange;}
```

```
        ul li:nth-child(3){background-color:Blue;}
        ul li:nth-child(4){background-color:silver;}
        ul li:nth-child(5){background-color:Purple;}
    </style>
</head>
<body>
    <ul>
        <li>1</li>
        <li>2</li>
        <li>3</li>
        <li>4</li>
        <li>5</li>
    </ul>
</body>
</html>
```

在浏览器预览效果如图6-14所示。

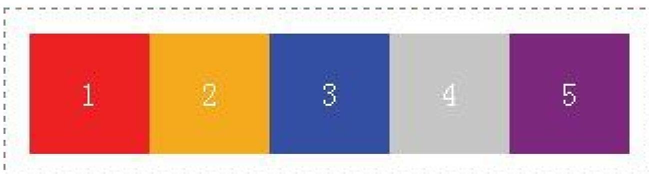


图6-14 “display:table-cell”自动平均划分元素

分析：

当我们将ul宽度改为400px时，在浏览器预览效果如图6-15所示。



图6-15 ul宽度改为400px时效果

从上面我们可以看到，ul元素宽度自动根据li元素个数进行平均划分，并不需要我们指定每一个li元素的宽度。

6.5 去除inline-block元素间距

我们先来看一个例子：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        *{padding:0;margin:0;}
        ul{list-style-type:none; }
        li
        {
            display:inline-block;
            width:60px;
            height:60px;
            line-height:60px;
            font-size:30px;
            text-align:center;
            color:White;
            background-color:Purple;
        }
    </style>
</head>
<body>
    <ul>
        <li>A</li>
        <li>B</li>
        <li>C</li>
    </ul>
</body>
</html>
```

在浏览器预览效果如图6-16所示。

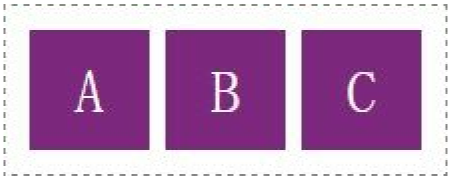


图6-16 inline-块元素的间距

在图中我们可以看到inline-block元素之间是有间距的。在实际开发中，这种间距有时会对我们的布局产生影响。大多数时候为了不影响布局，我们需要去除inline-block元素的间距。

在CSS中，我们可以使用“font-size:0”来去除inline-block元素的间距。

语法：

```
父元素{font-size:0;}
```

说明：

“font-size:0”是在inline-block元素的父元素添加的。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        *{padding:0;margin:0;}
        ul{list-style-type:none; font-size:0;}
        li
        {
            display:inline-block;
            width:60px;
            height:60px;
            line-height:60px;
            font-size:30px;
            text-align:center;
            color:White;
```

```
            background-color:Purple;
        }
    </style>
</head>
<body>
    <ul>
        <li>A</li>
        <li>B</li>
        <li>C</li>
    </ul>
</body>
</html>
```

在浏览器预览效果如图6-17所示。

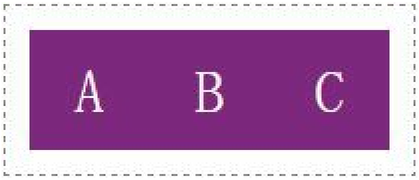


图6-17 “font-size:0”去除inline-块元素间距

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
</head>
<body>
    <img src="images/haizei1.png" alt=""/>
    <img src="images/haizei1.png" alt=""/>
    <img src="images/haizei1.png" alt=""/>
</body>
</html>
```

在浏览器预览效果如图6-18所示。



图6-18　图片之间的间距

当我们在CSS中添加“body{font-size:0}”后，在浏览器预览效果如图6-19所示。

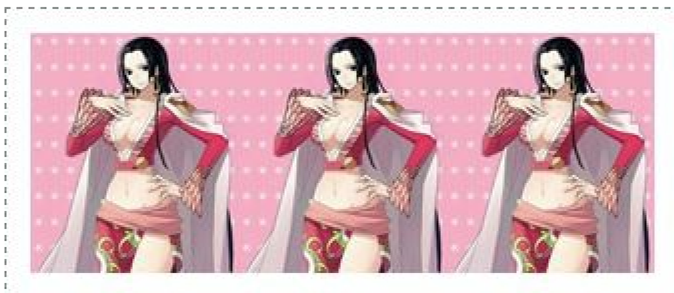


图6-19　“font-size:0”去除图片间距

分析：

由于img元素也是inline-block元素，因此我们也能使用“font-size:0”来去除图片之间的间距。

如果想要去除inline-block元素的间距，除了“font-size:0”，还可以使用负margin、letter-spacing、word-spacing等方法来实现。不过其他这几种方法或多或少会有其他问题，而且太多方法也容易增加记忆负担。因此，我们只需要掌握“font-size:0”这一个方法就行了。

第7章　文本效果

7.1　文本效果简介

在CSS入门阶段，我们学习了以上文本样式属性。为了让大家有一个循序渐进的学习过程，在入门时我们只是介绍一下基本的语法，并没有深入学习高级技巧。对于这些基本语法，可以查看《Web前端开发精品课HTML和CSS基础教程》。表7-1为CSS文本属性及其说明。

表**7-1 CSS**文本属性

属　　性	说　　明
text-decoration	下划线、删除线、顶划线
text-transform	文本大小写
font-variant	将英文文本转换为“小型”大写字母
text-indent	段落首行缩进
text-align	文本水平对齐
vertical-align	文本垂直对齐
line-height	行高

letter-spacing	字距
word-spacing	词距

text-decoration、text-transform、letter-spacing等属性相对比较简单，没多少东西可以研究。这一章带大家深入学习以下四个属性。

（1）text-indent。

（2）text-align。

（3）line-height。

（4）vertical-align。

7.2 深入text-indent

我们知道，在CSS中可以使用text-indent属性来定义段落的首行缩进。在之前“CSS单位”一节中，我们学习了关于text-indent的一个技巧：使用“text-indent:2em;”来实现段落的首行缩进。在这一节，我们来深入学习text-indent的另外一个技巧。

有过开发经验的小伙伴们，或多或少会见过“text-indent:-9999px;”这种写法。“text-indent:-9999px;”一般用于LOGO部分。在搜索引擎优化中，h1是非常重要的标签。一般情况下，我们都是把网站的LOGO图片放到h1标签中。不过我们都知道，搜索引擎无法识别图片，只能识别文字。为了更好地优化SEO，那该怎么做呢？

有一个很好的解决方法就是：指定h1元素的长宽与LOGO图片的长宽一样，然后定义h1的背景图片（background-image）为LOGO图片。也就是说，我们使用LOGO图片作为h1标签的背景图片，然后使用“text-indent:-9999px;”来隐藏h1的文字内容。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        h1
        {
            width:300px;
            height:100px;
            background-image:url("images/logo.jpg");
            text-indent:-9999px;
        }
    </style>
</head>
<body>
    <h1>绿叶学习网</h1>
```

```
</body>
</html>
```

在浏览器预览效果如图7-1所示。

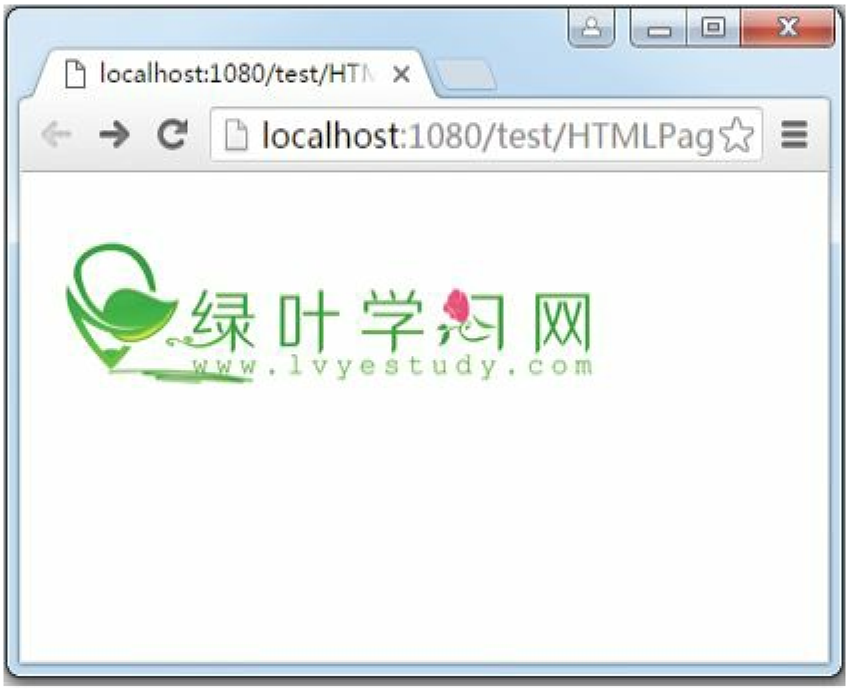


图7-1　“text-indent:-9999px;”来隐藏h1的文字内容

分析：

在这个例子中，使用“text-indent:-9999px;”隐藏了h1的内容。如果我们没有使用“text-indent:-9999px;”，得到的效果如图7-2所示。

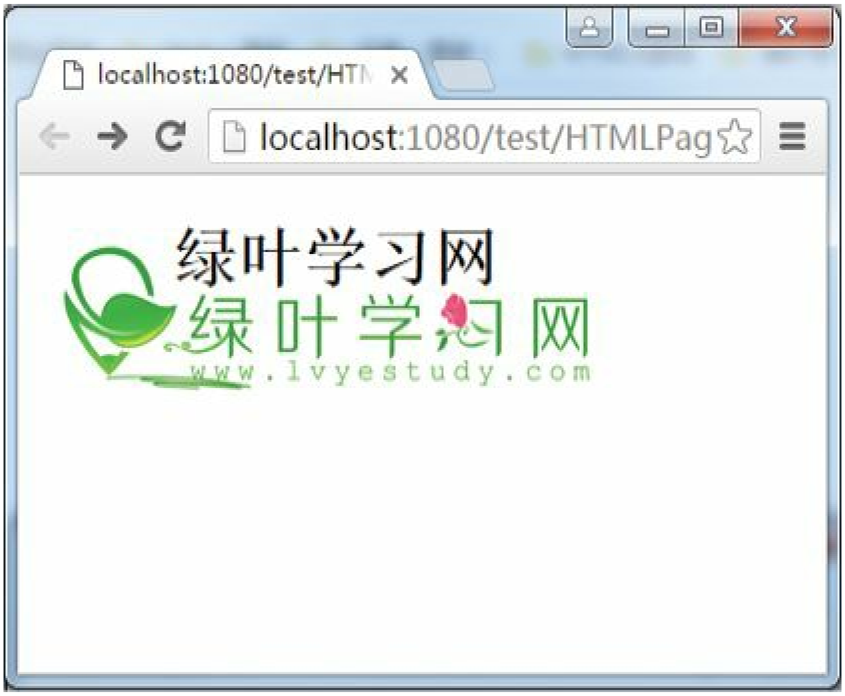


图7-2　没有使用“text-indent:-9999px;”时的效果

除了“text-indent:-9999px;”，我们还可以使用“display:none;”来隐藏文字，以便达到预期效果，实现代码如下：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        h1
        {
            width:300px;
            height:100px;
            background-image:url("images/logo.jpg");
        }
        h1 span{display:none;}
    </style>
</head>
<body>
    <h1><span>绿叶学习网</span></h1>
</body>
</html>
```

在浏览器预览效果如图7-3所示。

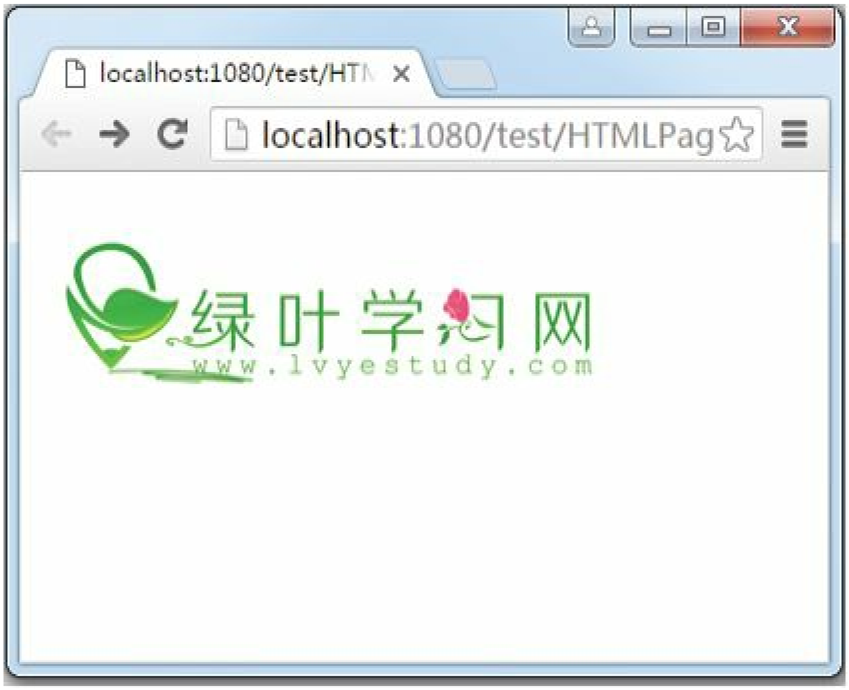


图7-3　“display:none”来隐藏h1的文字内容

分析：

其实页面效果是达到了，但是搜索引擎却不吃这一套！为什么呢？对于使用“display:none;”来隐藏的文字，搜索引擎一般都把这些文字当做垃圾信息而忽略，此时h1的权重会丢失。所以我们不建议使用这种方式。

对于“text-indent:-9999px”这个技巧，大家可以去查看一下绿叶学习网、w3cschool首页以及其他网站LOGO部分的源代码。实际上，这个技巧几乎在所有网站中都能用到，大家一定要认真掌握。

当然大家想要更好地理解这个技巧，也需要一定搜索引擎优化的知识。搜索引擎优化（即SEO）也是前端工程师必备知识之一。

【疑问】

为什么定义text-indent为-9999px，而不是-999px、-99px呢？

一般电脑屏幕宽度有为1024px、1366px等，之所以定义为-9999px，那是为了让文字的缩进足够大，就算是大分辨率电脑下也看不到文字，因为没有哪台电脑的屏幕宽度大于9999px。如果你定义text-indent为-999px或者-99px，那么缩进的文字还是有可能会出现在浏览器窗口内。

7.3　深入text-align

在CSS中，我们可以使用text-align属性来控制文本以及图片的对齐方式。text-align属性取值如表7-2所示。

表**7-2 text-align**属性取值

属　性　值	说　　明
left	左对齐
right	右对齐
center	居中

text-align还有一个属性值为justify（两端对齐），不过由于这个属性值本身会产生一些问题，所以在实际开发中比较少用。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        div
        {
            width:240px;
            height:100px;
            border:1px solid gray;
            text-align:center;
        }
    </style>
```

```
</head>
<body>
    <div>
        <img src="images/haizei2.png" alt=""/>
    </div>
    <div>海贼王娜美</div>
</body>
</html>
```

在浏览器预览效果如图7-4所示。



图7-4　text-align作用于文字和图片

分析：

从这个例子我们可以看出，text-align属性对文字和图片都起作用。

7.3.1　text-align起作用的元素

有人问，text-align属性是不是就只对文字和图片起作用呢？答案是否定的。一个比较准确的说法是：text-align对文字、inline元素（行内元素）以及inline-block元素（行内块元素）起作用，但对块元素不起作用。其中，img元素属于inline-block元素。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        #outer-box
        {
            width:120px;
            height:80px;
            border:1px solid gray;
            text-align:center;
        }
        #inner-box
        {
            width:50px;
            height:50px;
            background-color:Orange;
        }
    </style>
</head>
<body>
    <div id="outer-box">
        <div id="inner-box"></div>
    </div>
</body>
</html>
```

在浏览器预览效果如图7-5所示。

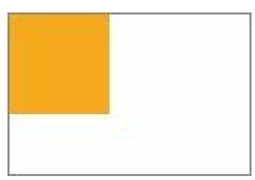


图7-5　text-align属性对div不起作用

分析：

我们都知道div是块元素，如果想要使用水平居中div元素，可以先使用“display:inline-block”来将div转化为inline-block元素，然后再使用“text-align:center”。当然，在后面“水平居中”一节我们会详细给大家

介绍更多关于水平居中元素的技巧。

7.3.2 “text-align:center;”与“margin:0 auto;”的区别

在页面水平居中实现中，“text-align:center;”与“margin:0 auto;”是最常见的两种水平居中方式。不过这两者也有着本质的区别。

（1）“text-align:center;”实现的是文字、inline元素以及inline-block元素的水平居中。

（2）“margin:0 auto;”实现的是块元素的水平居中。

（3）“text-align:center;”在父元素中定义，“margin:0 auto;”在当前元素中定义。

7.4　深入line-height

在CSS中，我们可以使用line-height属性来控制文本的行高。很多书称line-height为行间距，其实这是非常不严谨的叫法。行高，顾名思义就是“一行的高度”，而行间距指的是“两行文本之间的距离”，两者是完全不一样的概念。这一节，我们来深入学习一下line-height属性。

7.4.1　line-height的定义

在CSS中，line-height还有一个更加准确的定义：两行文字基线之间的距离。

1．顶线、中线、基线、底线

我们都用过英文簿，英文簿每一行都有4条线，这4条线分别是：顶线、中线、基线和底线，如图7-6所示。



图7-6　顶线、中线、基线、底线

在CSS中，每一行文字可以看成一个“行盒子”，而每一个行盒子都有4条线：顶线、中线、基线和底线。没错，这四条线跟英文簿中的四条线是一样的道理。

此外，vertical-align属性中的top、middle、baseline、bottom这四个属性值分别对应的就是：顶线、中线、基线、底线。

注意一下，基线并不是行盒子中最下面的线，而是倒数第二条线。由此我们很清楚地知道line-height究竟指的是什么。

2．行高、行距与半行距

（1）行高。

行高（即line-height），指的是“两行基线之间的垂直距离”，如图7-7所示。

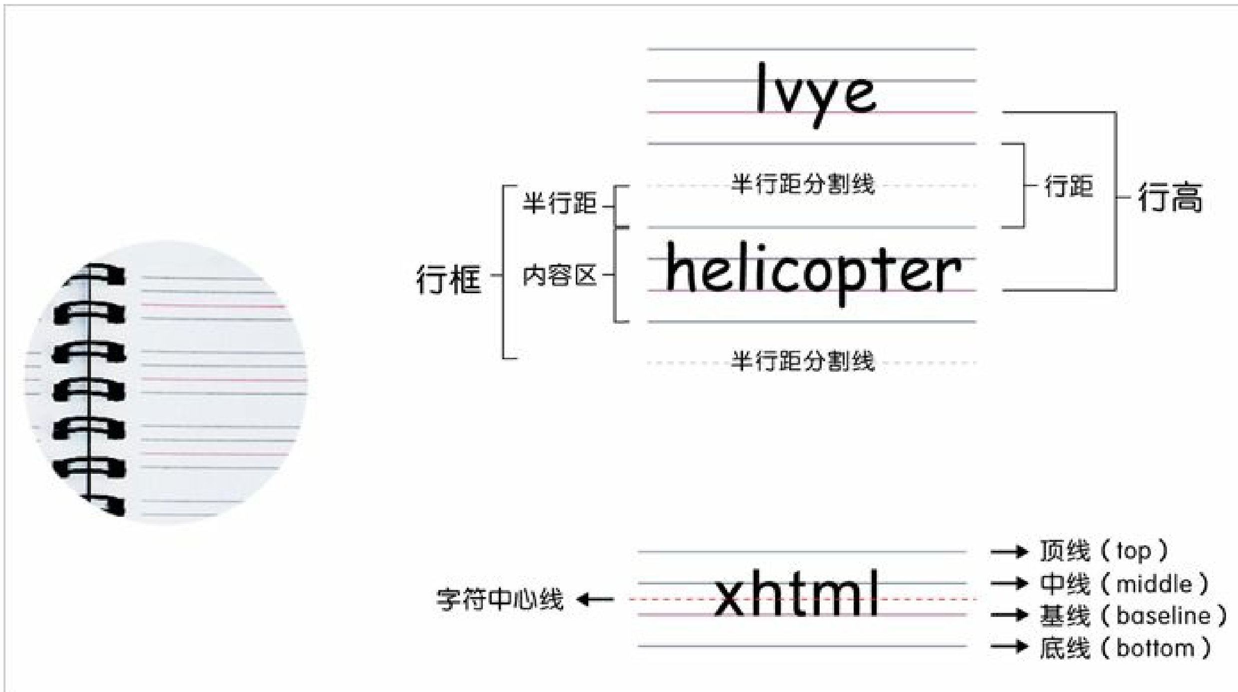


图7-7　line-height分析图

有些小伙伴就会问了：为什么W3C要这样去定义line-height呢？直接定义line-height为两条底线之间的距离岂不是更好理解么？规则这东西嘛，都是官方定义的，我们只需要去遵循就行了。这就跟我们过马路一样，没必要纠结为什么是“绿灯走红灯停”，而不是“红灯走绿灯停”。

（2）行距。

行距，指的是上一行的底线到下一行的顶线的垂直距离。也就是两行文字之间的空隙。

（3）半行距。

半行距，很好理解，指的是行距的一半。

为什么要搞一个半行距出来呢？其实这就是为了引出在下面所提到

的“行框（inline box）”。

3．内容区与行框

（1）内容区。

内容区，指的是行盒子顶线到底线之间的垂直距离。

（2）行框。

行框，指的是两行文字“行半距分割线”之间的垂直距离。

7.4.2　line-height取值

1．height和line-height

line-height有默认值，当没有定义line-height属性时，浏览器就会采用默认的line-height值。

一行文字的高度是由line-height决定，而不是由height决定的。例如在p标签中，一个p标签的文字可以有很多行，其中line-height定义的是一行文字的高度，而height定义的是整个段落的高度（p标签的高度）。

在CSS中，我们可以定义height和line-height这两个属性值相等，从而来实现单行文字的垂直居中。这是经常使用到的一个技巧，希望大家记住。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title></title>
    <style type="text/css">
       div
       {
```

```
            width:240px;
            height:60px;
            border:1px solid gray;
            font-size:12px;
            text-align:center;
        }
        #div1{line-height:20px;}
        #div2{line-height:40px;}
        #div3{line-height:60px;}
    </style>
</head>
<body>
    <div id="div1">height为50px，line-height为20px</div>
    <div id="div2">height为50px，line-height为40px</div>
    <div id="div3">height为50px，line-height为60px</div>
</body>
</html>
```

在浏览器预览效果如图7-8所示。

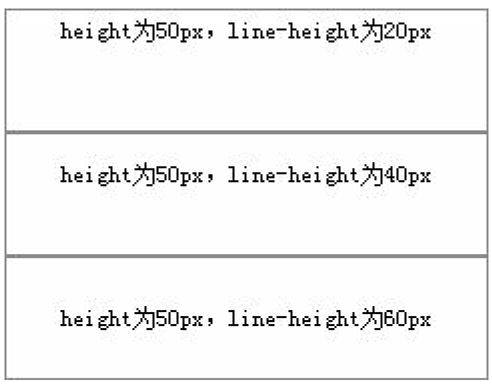


图7-8　单行文字垂直居中

分析：

为什么定义height和line-height这两个属性值相等，就可以实现单行文字的垂直居中呢？从上面这个例子，我们可以很直观而感性地认知。

2．line-height取值为百分比值、em值

当line-height值为百分比值或者em值时，当前元素的行高是相对于父元素的font-size值来计算的。计算公式如下：

```
line-height = （父元素font-size）×（百分比）
line-height = （父元素font-size）×（em值）
```

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title></title>
     <style type="text/css">
        body{font-size:30px;}
        #outer-box
        {
            /*父元素行高：30px×150%=45px*/
            line-height:150%;
            background-color:Red;
            color:White;
        }
        #inner-box
        {
            /*子元素行高：30px×150%=45px（继承父元素的line-height）*/
            font-size:20px;
            background-color:Purple;
            color:White;
        }
     </style>
</head>
<body>
     <div id="outer-box">这是父元素
           <div id="inner-box">这是子元素</div>
     </div>
</body>
</html>
```

在浏览器预览效果如图7-9所示。



图7-9　line-height取值为百分比值

分析：

在上面这段代码中，#outer-box的行高为30px× 1.5=45px。由于line-height具有继承性，当line-height取值为百分比时，会直接继承父元素的line-height（除非自己指定line-height）。

3．line-height取值为无单位数字

line-height还支持无单位数字的属性取值，在CSS中也只有line-height属性具有这个特点。当line-height值为无单位数字时，实际的行高是相对于当前元素的font-size值来计算的。计算公式如下：

```
line-height = （当前元素的font-size）×（无单位数字）
```

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title></title>
    <style type="text/css">
        body{font-size:30px;}
        #outer-box
        {
            /*父元素行高：30px×1.5=45px*/
            line-height:1.5;
```

```
            background-color:Red;
            color:White;
        }
        #inner-box
        {
            /*子元素行高：20px×1.5=30px（继承父元素的系数）*/
            font-size:20px;
            background-color:Purple;
            color:White;
        }
    </style>
</head>
<body>
    <div id="outer-box">这是父元素
        <div id="inner-box">这是子元素</div>
    </div>
</body>
</html>
```

在浏览器预览效果如图7-10所示。



图7-10　line-height取值为无单位数字

分析：

在上面这段代码中，#outer-box的行高为30px× 1.5=45px，#inner-box的行高为20px×1.5=30px。也就是说当line-height取值为无单位数字时，该无单位数字可以理解为一个系数。子元素继承的是父元素的系数，不会直接继承父元素的line-height。

7.5　深入vertical-align

vertical-align，很多人对这个属性感到很陌生，也不知道怎么去使用，其实这都是由于没有深入属性的本质所导致的。vertical-align属性非常复杂，但是也相当强大。这一节给大家介绍vertical-align属性最实用的技巧，而对于不常用的东西则不进行深入研究。

W3C官方对vertical-align属性的定义有四个方面。

（1）vertical-align属性用于定义“周围的文字、inline元素以及inline-block元素”相对于该元素基线的垂直对齐方式。这里的“该元素”指的是被定义了vertical-align属性的元素。

（2）在表格单元格中，vertical-align属性可以定义单元格td元素中内容的对齐方式。td元素是table-cell元素，也就是说vertical-align属性对table-cell类型元素有效。

（3）vertical-align属性对inline元素、inline-block元素和table-cell元素有效， 对块元素无效。

（4）vertical-align属性允许指定负长度值（如-2px）和百分比值（如50%）。

从上一节我们知道，每一行文字可以看成一个行盒子，其实每一个inline-block元素也可以看成一个行盒子。其中，每一个行盒子都有四条线：顶线、中线、基线和底线，如图7-11所示。这四条线跟英文簿中的四条线是相似的。



图7-11　顶线、中线、基线、底线

vertical-align属性中的基线跟line-height属性中的基线是一样的道

理。在CSS中，vertical-align属性最见的属性值有四个：top（顶部对齐）、middle（中部对齐）、baseline（基线对齐）、bottom（底部对齐）。

7.5.1　vertical-align属性取值

vertical-align属性取值有三种情况：负值、百分比和关键字。

1．负值

vertical-align属性取值可以为负值，例如“vertical-align:-2px”表示元素相对于基线向下偏移2px。此方法常常用于解决单选框或复选框与文字垂直对齐的问题。对于这个技巧，我们在下一章“表单效果”会详细介绍。

2．百分比

vertical-align属性取值可以为百分比，这个百分比是相对于当前元素所继承的line-height属性值决定的。

举个例子，对于“vertical-align:50%”，假如当前元素所继承的line-height为20px，则“vertical-align:50%”实际上等价于“vertical-align:10px”。其中， vertical-align:10px”表示元素相对于基线向上偏移10px。

3．关键字

vertical属性取值可以为关键字，取值如表7-3所示。

表**7-3 vertical**属性取值为**“**关键字**”**

取　　值	说　　明
top	顶部对齐

middle	中部对齐
baseline	基线对齐
bottom	底部对齐

除了以上这些，vertical-align属性取值还可以为text-top、text-bottom、super、sub等关键字。不过其他关键字在实际开发中很少用得上，因此我们只需要掌握top、middle、baseline、bottom这四个属性值就行。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>vertical-align属性</title>
    <style type="text/css">
        img{width:80px;height:80px;}
        #img1{vertical-align:top;}
        #img2{vertical-align:middle;}
        #img3{vertical-align:bottom;}
        #img4{vertical-align:baseline;}
    </style>
</head>
<body>
    绿叶学习网<img id="img1" src="images/girl.png" alt=""/>绿叶学习网（<stron
    <hr/>
    绿叶学习网<img id="img2" src="images/girl.png" alt=""/>绿叶学习网（<stron
    <hr/>
    绿叶学习网<img id="img3" src="images/girl.png" alt=""/>绿叶学习网（<stron
    <hr/>
    绿叶学习网<img id="img4" src="images/girl.png" alt=""/>绿叶学习网（<stron
</body>
</html>
```

在浏览器预览效果如图7-12所示。



图7-12　vertical-align各个属性值效果

分析：

根据W3C的定义，vertical-align属性用于定义周围文字、inline元素或inline-block元素的基线相对于该元素的基线的垂直对齐方式。在这个例子中，vertical-align属性定义了周围的文字相对于img元素基线的垂直对齐方式。

此外，“vertical-align:baseline”和“vertical-align:bottom”是有区别的，请仔细观察预览效果。

7.5.2　vertical-align属性应用

我们从以下三个方面来介绍一下vertical-align属性的使用情况：inline-block元素、inline元素和块元素、table-cell元素。

1．inline元素和inline-块元素

在HTML中，常见的inline-block元素有两个：img元素和input元素。对于这两个inline-block元素，我们一定要记住。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        strong
        {
            font-size:40px;
            border:1px solid red;
        }
        span{font-size:12px;}
    </style>
</head>
<body>
    <span>绿叶学习网</span><strong>绿叶学习网</strong><span>绿叶学习网</span>
</body>
</html>
```

在浏览器预览效果如图7-13所示。

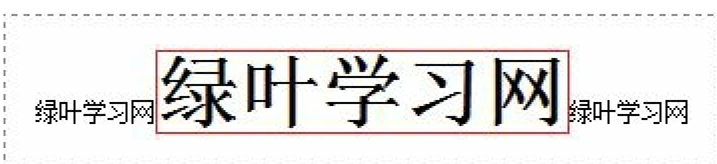


图7-13　未设置“vertical-align:middle”的strong元素

分析：

当我们在CSS中为strong元素添加“vertical-align:center;”之后，浏览器预览效果如图7-14所示。



图7-14　设置“vertical-align:middle”后的strong元素

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title></title>
    <style type="text/css">
        img{vertical-align:middle;}
    </style>
</head>
<body>
    <div>绿叶学习网<img src="images/girl.png" alt=""/>绿叶学习网（<strong>mid
</body>
</html>
```

在浏览器预览效果如图7-15所示。



图7-15　设置了“vertical-align:middle”的图片

2．块元素

vertical-align属性对inline元素、inline-block元素和table-cell元素有效，对块元素无效。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title></title>
    <style type="text/css">
        div
        {
            vertical-align:middle;
            width:120px;
            height:120px;
            border:1px solid gray;
        }
    </style>
</head>
<body>
    <div><img src="images/girl.png" alt=""/></div>
</body>
</html>
```

在浏览器预览效果如图7-16所示。

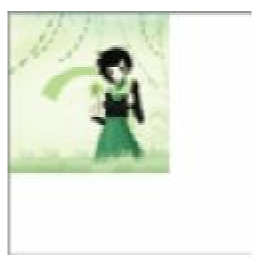


图7-16　设置“vertical-align:middle”的div元素

分析：

div是块元素，所以vertical-align属性对其无效。如果想要在div中实现图片的垂直居中，我们可以先为div定义display:table-cell，也就是将块元素转化为table-cell元素（表格单元格），然后再使用vertical-align:middle就可以实现了。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title></title>
    <style type="text/css">
        div
        {
            display:table-cell;
            vertical-align:middle;
            width:120px;
            height:120px;
            border:1px solid gray;
        }
    </style>
</head>
<body>
    <div><img src="images/girl.png" alt=""/></div>
</body>
</html>
```

在浏览器预览效果如图7-17所示。

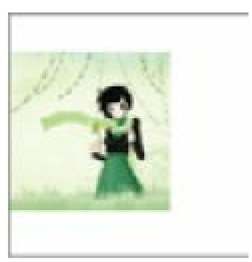


图7-17 “vertical-align:middle”实现图片垂直居中于div元素

分析：

在div中实现图片垂直居中是很常见的技巧，我们在后续章节会给大家介绍更多垂直居中的技巧。

3．table-cell元素

W3C定义，在表格单元格中，vertical-align属性可以定义单元格中内容的对齐方式。也就是说vertical-align属性对table-cell类型元素有效。

这里要注意一下，table-cell元素跟inline、inline-block元素使用vertical-align是有很大区别的。

（1）inline元素和inline-block元素的vertical-align是针对周围的元素来说的，vertical定义的是周围元素相对于当前元素的对齐方式。

（2）table-cell元素的vertical-align属性是针对自身而言。vertical-align定义的是内部子元素相对于自身的对齐方式。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title></title>
    <style type="text/css">
        td
        {
            width:120px;
            height:120px;
            border:1px solid gray;
            vertical-align:middle;
        }
    </style>
</head>
<body>
    <table>
        <tr>
            <td><img src="images/girl.png" alt=""/></td>
            <td><img src="images/girl.png" alt=""/></td>
            <td><img src="images/girl.png" alt=""/></td>
        </tr>
    </table>
</body>
</html>
```

在浏览器预览效果如图7-18所示。

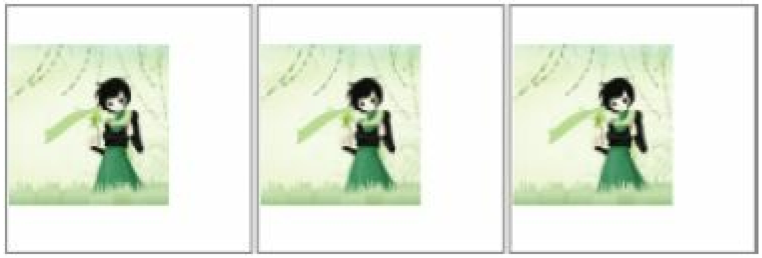


图7-18　设置了“vertical-align:middle”的td元素

【总结】

这一章学习到的知识，估计已经狠狠地给那些天天自诩“精通CSS”的同学几巴掌了。原来，CSS也是如此博大精深，并非我们想象中那么简单。对于HTML、CSS和JavaScript这三大核心技术，还是希望大家能够踏踏实实地深入研究，这样我们的前端之路才有可能走得更远。

第8章　表单效果

8.1　表单效果简介

表单，在实际开发中十分常见，几乎每一个网站都会涉及表单。一个表单是否美观，对用户体验来说非常重要。毕竟在一个站点中，表单是用户注册登录经常用到的东西。别看一个表单结构那么简单，有时候使用CSS操作起来是一件很头疼的事情。如果让我们来控制表单的外观，估计不少小伙伴会出现各种各样的问题，例如文本框与文字不能对齐，复选框与文字没有垂直居中，图8-1为绿叶学习网中的表单。



图8-1　绿叶学习网中的表单

这一章我们从以下三个方面来深入学习一下表单开发技巧。

（1）深入radio和checkbox。

（2）深入textarea。

（3）表单对齐。

在这一章我们只介绍表单中的CSS技巧，对于表单的动态操作，如

文本框聚焦失焦、单复选框全选等，可以关注绿叶学习官网的JavaScript教程或jQuery教程。

8.2　深入radio和checkbox

radio指的是单选框，而checkbox指的是复选框。对于radio和checkbox，我们主要来学习一个方面就行了：单选框复选框与文字垂直居中对齐。

现在大部分网站的主流字体大小都是12px（如图8-2所示）或者14px（如图8-3所示）。在表单开发中，无论是12px，还是14px，文字与单选框或复选框都是不对齐的，感觉特别难看，用户体验比较差。



图8-2　字体大小为12px



图8-3　字体大小为14px

之前我们接触过vertical-align属性，单选框或复选框与文字默认情况下是以vertical-align:baseline的方式对齐的，这也是导致单选框或复选

框与文字在垂直方向不居中对齐的原因。因此，我们可以使用vertical-align属性来解决。

语法：

```
vertical-align:像素值;
```

说明：

我们分两种情况考虑。

（1）当文字大小为12px时，我们给单选框或复选框定义“vertical-align:-3px”即可解决对齐问题。

（2）当文字大小为14px时，我们给单选框或复选框定义“vertical-align:-2px”即可解决对齐问题。

其中，“vertical-align:-3px”表示元素相对于基线向下偏移3px。对于基线的概念，我们在“7.4　深入line-height属性”一节详细给大家介绍了。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        /*文字大小为12px*/
        #p1{font-size:12px;}
        #p1 input{vertical-align:-3px;}
```

```
        /*文字大小为14px*/
        #p2{font-size:14px;}
        #p2 input{vertical-align:-2px;}
    </style>
</head>
<body>
    <p id="p1">
        <input id="rdo1" type="radio" /><label for="rdo1">单选框</label>
        <input id="cbk1" type="checkbox" /><label for="cbk1">复选框</label>
    </p>
    <p id="p2">
        <input id="rdo2" type="radio" /><label for="rdo2">单选框</label>
        <input id="cbk2" type="checkbox" /><label for="cbk2">复选框</label>
    </p>
</body>
</html>
```

在浏览器预览效果如图8-4所示。

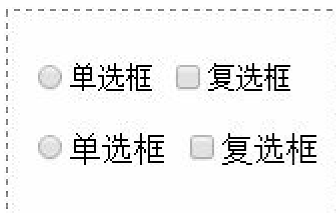


图8-4 vertical-align实现单选框和复选框与文字对齐

分析：

上面是文字大小为12px和14px时的解决方法，如果你的页面文字大小为15px、16px等，办法也是一样的。我们只需要稍微调整一下vertical-align属性的数值，直到效果满意即可，很简单。

8.3　深入textarea

textarea指的是文本域。对于textarea，有两个方面需要我们深入学习。

（1）固定大小，禁用拖动。

（2）在chrome、Firefox和IE实现相同的外观。

8.3.1　固定大小，禁用拖动

在主流浏览器中，如果我们使用Chrome或Firefox浏览器，会发现textarea元素右下角有一个小三角，如图8-5所示。当用鼠标拖拽小三角的时候，我们会发现textarea可以放大或缩小。Chrome浏览器和Firefox浏览器都有这个功能，但是IE浏览器是没有的，如图8-6所示。

其实这个功能是浏览器方便用户而添加的，因为前端开发人员在设计页面的时候，为了页面布局，一般会给textarea设定一个固定的长度。但是有些用户往往会认为textarea太小或者太大了。有了小三角拖拽的功能，用户就可以自己调整合适的大小了。



图8-5　Chrome浏览器下的textarea



图8-6　IE浏览器下的textarea

但是我们知道，如果用户过分拖动textarea的大小会影响页面布局，使得页面不美观。因此在实际开发中我们往往都是设定固定大小或者禁止拖动。

1．固定大小

在CSS中，我们可以使用min-width和min-height来定义textarea的最小宽度和最大高度，也可以使用max-width和max-height来定义textarea的最大宽度和最大高度。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        textarea
        {
            width:100px;
            height:80px;
            max-width:200px;
            max-height:160px;
        }
    </style>
</head>
<body>
    <textarea></textarea>
</body>
</html>
```

在浏览器预览效果如图8-7所示。



图8-7　textarea固定大小

分析：

一般情况下对于textarea元素来说，min-width和min-height用得较少，max-width和max-height用得较多。max-width和max-height可以控制textarea的最大宽度和最大高度，使得用户在拖动小三角的同时，不会破坏原来的布局。

2. 禁止拖动

如果我们想要彻底禁止用户拖动小三角来改变textarea元素大小，可以使用CSS中的resize属性来实现。

语法：

```
resize:none;
```

说明：

“resize:none”表示禁用textarea元素的拖动功能，此时textarea元素右下方的小三角会消失。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        textarea
        {
            width:100px;
            height:80px;
            resize:none;
```

```
        }
    </style>
</head>
<body>
    <textarea></textarea>
</body>
</html>
```

在浏览器预览效果如图8-8所示。



图8-8　textarea禁止拖动

分析：

如果想要禁止用户拖动textarea，除了可以用“resize:none”之外，还有一种方法：将max-width与width定义相同值，并且将max-height与height定义相同值。不过在实际开发中，还是推荐使用“resize:none”这种实现方式比较简单。

8.3.2　在chrome、Firefox和IE实现相同的外观

textarea元素有cols和rows两个属性，其中cols用来控制列数（也就是每行文字个数），rows用来控制行数。但是如果使用cols和rows属性来控制textarea的外观大小，我们会发现以下两个特点。

（1）Chrome（或Firefox）和IE下，每行字数和文字的列数是不相同的。

（2）默认情况下，IE是有滚动条的，而Chrome（或Firefox）是没有滚动条的（从图8-5和图8-6可以看出）。

那在实际开发中，怎么使得textarea在Chrome（或Firefox）和IE下

具有相同的外观效果呢？其实很简单。

（1）使用CSS的width和height来定义textarea的大小。

（2）使用“overflow:auto”来定义textarea的滚动条自适应。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        textarea
        {
            width:100px;
            height:80px;
            overflow:auto;
            resize:none;
        }
    </style>
</head>
<body>
    <textarea></textarea>
</body>
</html>
```

在浏览器预览效果如图8-9所示。

图8-9　在Chrome、Firefox和IE实现相同的textarea外观

分析：

对于上面这个例子代码，我们自行在各个浏览器中查看，textarea元素的外观都是一样的。

8.4　表单对齐

在表单操作中，我们经常要实现如下图8-10所示的对齐效果，左边是文字，右边是表单元素。表单元素排列很整齐，非常美观。表单对齐，是表单操作中必然碰到的一件事。看似很简单，如果不懂得技巧，实现起来并不一定很容易。



图8-10　百度的注册表单

为了实现对齐，不少初学者都是直接对input或label元素进行逐个设置padding或者margin来慢慢调，这样使得CSS代码非常多，也难以维护。

很多大型网站包括百度、京东、腾讯等都是采用如下方法来实现。

（1）每一行表单分为左栏加若干右栏。所有行的左栏长度相等，所有行的右栏所有盒子长度之和相等。左栏一般是一个label，右栏是若干个文本框。

（2）所有左栏盒子和右栏盒子都设置为左浮动。

（3）左栏text-align属性定义为right，使得文字右对齐。

（4）最重要一点，每一行中左栏长度和右栏所有盒子的总长度之和等于行宽。这里的盒子是包括width、padding、border和margin的。

我们还是先看一个例子。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title></title>
    <style type="text/css">
        form
        {
            width:320px;
            font-family:Arial;
            font-size:14px;
            font-weight:bold;
        }
        /*清除每一个p中的浮动*/
        p{overflow:hidden;}
        label
        {
            float:left;
            width:60px;
            height:40px;
            line-height:40px;
            text-align:right;
            margin-right:10px;
        }
        input:not(#submit)
        {
            float:left;
            height:16px;
            padding:10px;
            border:1px solid silver;
        }
        #tel,#pwd
        {
            width:228px;
        }
        #verifyCode
        {
            width:118px;
            margin-right:10px;
        }
        #submit
        {
            width:100px;
            height:40px;
```

```
            border:1px solid gray;
            padding:0;
            background-color:#F1F1F1;
        }
    </style>
</head>
<body>
    <form action="index.html">
        <p>
            <label for="tel">手机号</label>
            <input id="tel" type="text"/>
        </p>
        <p>
            <label for="pwd">密码</label>
            <input id="pwd" type="password"/>
        </p>
        <p>
            <label for="verifyCode">验证码</label>
            <input id="verifyCode" type="text"/>
            <input id="submit" type="submit"/>
        </p>
    </form>
</body>
</html>
```

在浏览器预览效果如图8-11所示。



图8-11　表单对齐实例

分析：

在这里我们使用“overflow:hidden”来清除每一个p元素中的浮动。浏览器自带的调试工具可以很好地帮助我们查看各个盒子的长度情况，方便我们调试。对于Chrome浏览器，我们可以使用“Ctrl+Shift+I”组合键打开调试工具。而对于Firefox，我们可以使用Firebug工具来调试。这个实例的CSS样式过多，但是我们只需要关注长度的计算即可。

第一行中长度情况如图8-12所示，60px是左盒子的长度（width+border+padding），250px是右盒子的长度（width+border+padding）。因此在第一行总长度为60px+250px+10px = 320px，刚好等于行宽320px。其中10px是margin值。

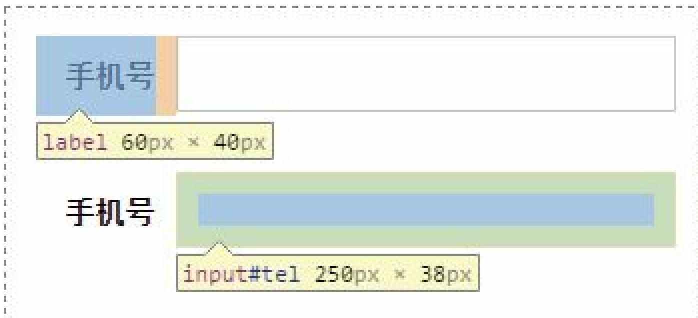


图8-12　第一行长度分析图

第三行中的长度情况如图8-13所示，60px是左盒子的长度（width+border+padding），140px是中间盒子的长度（width+border+padding），100px是右盒子的长度（width+border+padding）。因此第三行总长度为60px+140px+100px+10px+10px=320px，刚好等于行宽320px。其中10px是margin值。

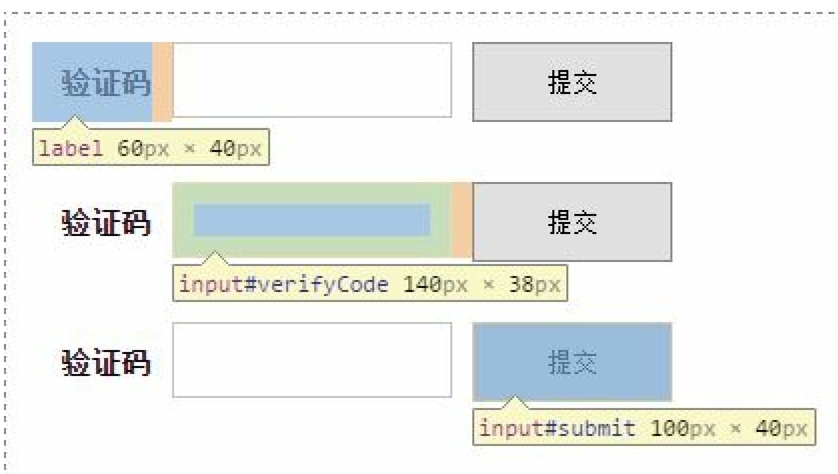


图8-13　第3行长度分析图

建议大家在本地编辑器中查看本书附带的源代码，并且结合浏览器调试工具进行学习。此外，在页面布局中如果碰到问题，建议大家多去查看大型网站源码中的解决方案，相信你能从中学到很多东西。

第9章　浮动布局

9.1　正常文档流

在学习浮动布局和定位布局之前，我们先来了解“正常文档流”和“脱离文档流”。了解这两个概念，是深入学习浮动布局和定位布局的理论前提。

9.1.1　正常文档流

什么叫“文档流”？简单来说，就是指元素在页面中出现的先后顺序。那什么叫“正常文档流”呢？

正常文档流，又称为“普通文档流”或“普通流”，也就是W3C标准所说的“normal flow”。我们先来看一下正常文档流的简单定义：“正常文档流，将一个页面从上到下分为一行一行，其中块元素独占一行，相邻行内元素在每一行中按照从左到右排列直到该行排满。”也就是说，正常文档流指的就是默认情况下页面元素的布局情况。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
</head>
<body>
    <div></div>
    <span></span><span></span>
```

```
    <p></p>
    <span></span><i></i>
    <img />
    <hr />
</body>
</html>
```

上面HTML代码的正常文档流如图9-1所示。

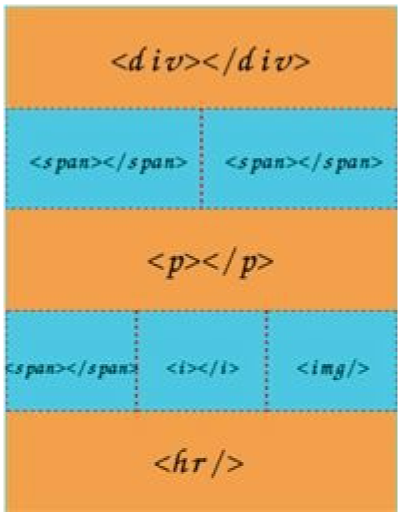


图9-1　HTML代码对应的正常文档流

分析：

由于div、p、hr都是块元素，因此独占一行。而span、i、img都是行内元素，因此如果两个行内元素相邻，就会位于同一行，并且从左到右排列，如图9-2所示。

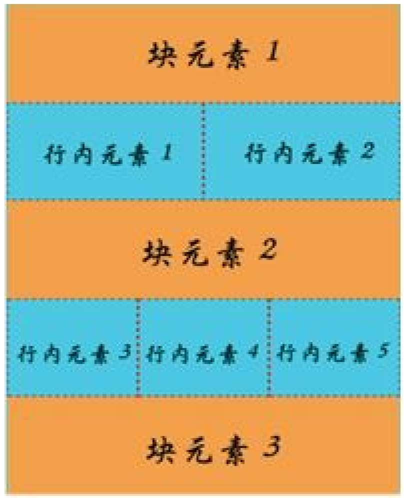


图9-2　正常文档流分析图

9.1.2　脱离文档流

脱离文档流，指的是脱离正常文档流。正常文档流就是我们没有使用浮动或者定位去改变的默认情况下的HTML文档结构。换一句话说，如果我们想要改变正常文档流，可以使用有两种方法：浮动和定位。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>CSS浮动float属性</title>
    <style type="text/css">
        /*定义父元素样式*/
        #father
        {
            width:300px;
            background-color:#0C6A9D;
            border:1px solid silver;
        }
        /*定义子元素样式*/
        #father div
        {
            padding:10px;
```

```
            margin:15px;
            border:2px dashed red;
            background-color:#FCD568;
        }
    </style>
</head>
<body>
    <div id="father">
        <div id="son1">box1</div>
        <div id="son2">box2</div>
        <div id="son3">box3</div>
    </div>
</body>
</html>
```

在浏览器预览效果如图9-3所示。

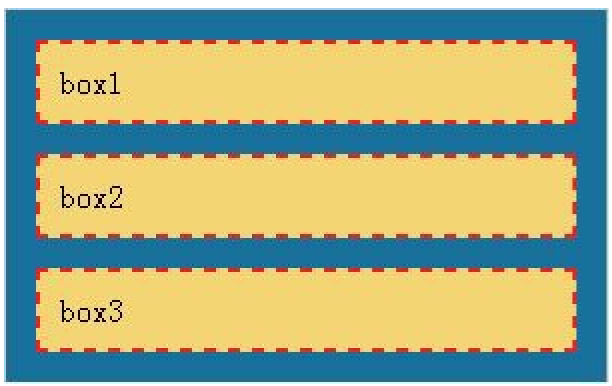


图9-3　正常文档流效果

分析：

上面定义了三个div元素。对于这个HTML来说，正常文档流指的就是从上到下依次显示这三个div元素。由于div是块元素，因此每个div元素独占一行。

1．设置浮动

当我们为第二、三个div元素设置左浮动时，在浏览器预览效果如

图9-4所示。

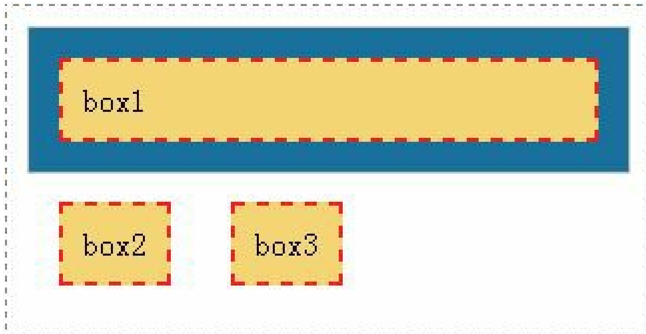


图9-4　浮动效果

正常文档流情况下，div是块元素会独占一行。但是由于设置了浮动，第二、三个div元素却是并列一行，并且跑到父元素之外，跟正常文档流不一样。也就是说，设置浮动使得元素脱离了正常文档流。

2．设置定位

当我们为第3个div元素设置绝对定位的时候，在浏览器预览效果如图9-5所示。

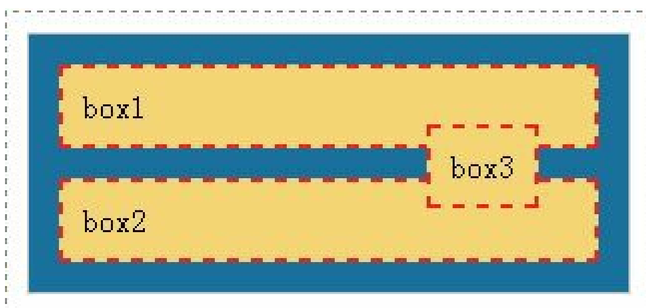


图9-5　定位效果

由于设置了定位，第三个div元素跑到父元素的上面去了。也就是说，设置了定位使得元素脱离了文档流。

“正常文档流”和“脱离文档流”比较抽象，建议大家结合一下后面介绍的“层叠上下文”这一节，估计就很容易理解了。

9.2　深入浮动

我们都知道在正常文档流的情况下，块元素都是独占一行的。如果想要使得两个或者多个块元素并排在同一行，这个时候可以考虑使用浮动，将块元素脱离正常文档流来实现。

浮动，可以使得元素移到左边或者右边，并且允许后面的文字或元素环绕着它。浮动，最常用于实现水平方向上的并排布局，例如两列布局、多列布局。也就是说，如果你想要实现两列并排或者多列并排效果时，可以考虑浮动，如图9-6所示。浮动一般用于实现水平方向的布局，而不是垂直方向上的布局。



图9-6　多列布局

float属性很简单，只有三个取值：left、right和none。但是浮动涉及的理论知识却非常多，其中包括：块元素和行内元素、CSS盒子模型、脱离文档流、BFC、层叠上下文。

对于浮动，具有以下两个最重要的特点。

（1）当一个元素定义了“float:left”或“float:right”时，不管这个元素之前是inline、inline-block或者其他类型，都会变成block类型。也就是说，浮动元素表现为块元素效果，可以定义width、height、padding和margin。这里特别要注意一下，我们可以使用margin-left或margin-right来定义浮动元素与其他元素之间的间距。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        span
        {
            float:left;
            width:50px;
            height:80px;
            border:1px solid gray;
            margin-left:10px;
        }
    </style>
</head>
<body>
    <span></span>
    <span></span>
    <span></span>
</body>
</html>
```

在浏览器预览效果如图9-7所示。

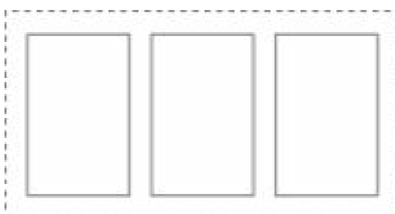


图9-7　浮动实现多列布局

分析：

span本身是inline元素，但是设置了浮动之后，就变成了块元素，并且可以设置width、height和margin等。

（2）当一个元素定义了“float:left”或“float:right”时，这个元素会脱离文档流，后面的元素会紧跟着填上空缺的位置。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        #wrapper
        {
            width:400px;
            height:200px;
            border:1px solid gray;
            padding:10px;
        }
        img{float:left;}
    </style>
</head>
<body>
    <div id="wrapper">
        <img src="images/ailianshuo.png" alt=""/>
        <div id="content">水陆草木之花，可爱者甚蕃。晋陶渊明独爱菊。自李唐来，世
    </div>
</body>
</html>
```

在浏览器预览效果如图9-8所示。

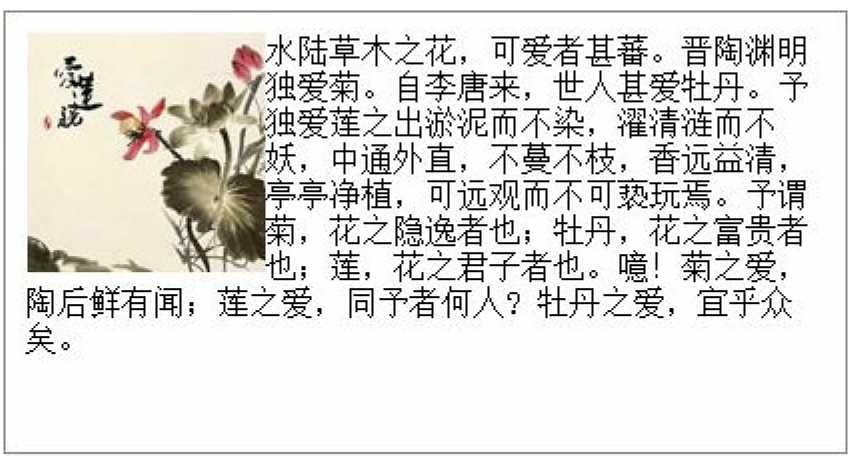


图9-8　浮动实现文字环绕

分析：

img元素设置了浮动，后面的p元素的内容会紧跟着填上父元素的空缺位置。

9.3 浮动的影响

这一节给大家介绍一下元素在设置了浮动之后带来的影响，以便让大家更加深刻地理解浮动。

（1）对自身的影响。

（2）对父元素的影响。

（3）对兄弟元素的影响。

（4）对子元素的影响。

9.3.1 对自身的影响

如果一个元素设置了浮动，则不管这个元素是什么类型，都会转化为块元素，也就是display属性值为block。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        strong
        {
            float:left;
            width:120px;
            height:60px;
            line-height:60px;
            border:1px solid gray;
            text-align:center;
        }
    </style>
</head>
```

```
<body>
    <strong>绿叶学习网</strong>
</body>
</html>
```

在浏览器预览效果如图9-9所示。



图9-9 浮动对自身的影响

分析：

strong元素是inline元素，但是在设置了浮动之后变成了块元素，并且可以设置width、height、padding和margin。

9.3.2 对父元素影响

如果一个元素设置了浮动，它会脱离正常文档流。如果浮动元素的高度height大于父元素的高度height，或者父元素没有定义高度height，此时浮动元素会脱离父元素。这就是我们常见的“父元素高度塌陷”。

造成父元素高度塌陷的原因在于，父元素的高度小于子元素的高度，或者父元素没有定义高度，父元素不能把子元素包裹起来。说白了，就是老爸管不住儿子，因此儿子离家出走了。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
```

```
    <style type="text/css">
        #wrapper
        {
            width:200px;
            border: 1px solid black;
        }
        #first,#second
        {
            width:80px;
            height:40px;
            border:1px solid red;
        }
        #first{float:left;}
        #second{float:right;}
    </style>
</head>
<body>
    <div id="wrapper">
        <div id="first"></div>
        <div id="second"></div>
    </div>
</body>
</html>
```

在浏览器预览效果如图9-10所示。

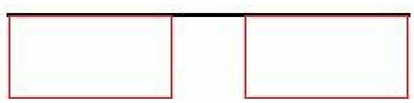


图9-10　浮动对父元素的影响

分析：

在这个例子中，由于父元素没有定义高度，因此父元素无法把子元素包裹起来，就会造成父元素高度塌陷。

如果我们为父元素定义高度，但是这个高度小于子元素高度（例如height:20px），在浏览器预览效果如图9-11所示。此时我们会发现，父元素无法把子元素包裹起来，仍然有父元素高度塌陷问题。

如果我们为父元素定义高度，但是这个高度大于子元素高度（例如height:60px），在浏览器预览效果如图9-12所示。此时我们会发现，父元素把子元素包裹起来，因此不会有父元素高度塌陷问题。

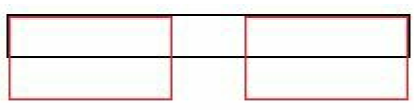


图9-11　设置父元素小于子元素高度

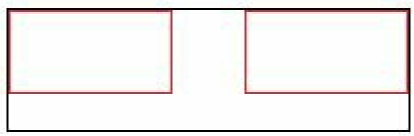


图9-12　设置父元素高度大于子元素

9.3.3　对兄弟元素的影响

1．兄弟元素是浮动元素

当一个浮动元素，它的兄弟元素也是浮动元素时，我们分两种情况来探讨：① 同一方向的兄弟元素；② 相反方向的兄弟元素。

（1）同一方向的兄弟元素。

当一个浮动元素碰到同一个方向的兄弟元素时，这些元素会从左到右、从上到下，一个接着一个紧挨着排列。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        body{text-align:center;}
        #main
```

```
        {
            text-align:left;
            display:inline-block;
            padding:12px;
            border:1px dashed gray;
        }
        #wrapper
        {
            width:240px;
            height:20px;
            border:1px solid black;
        }
        #first, #second
        {
            float:left;
            width:60px;
            height:60px;
            border:1px solid gray;
            margin-top:10px;
            margin-left:10px;
            margin-right:10px;
            background-color: #F4F6F4;
        }
    </style>
</head>
<body>
    <div id="main">
    <div id="wrapper">
        <div id="first">1</div>
        <div id="second">2</div>
    </div>
</div>
</body>
</html>
```

在浏览器预览效果如图9-13所示。

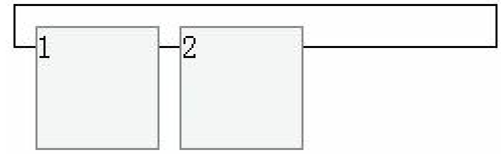


图9-13　同一方向的兄弟元素（左浮动）

分析：

当这两个div元素同时设置为右浮动时，在浏览器预览效果如图9-14所示。

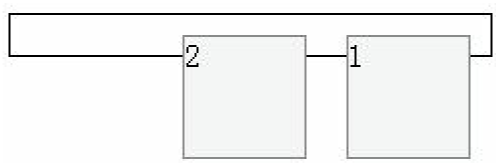


图9-14　同一方向的兄弟元素（右浮动）

（2）相反方向的兄弟元素。

当一个浮动元素碰到同一个方向的兄弟元素时，这两个元素会移向两边（如果父元素宽度足够的话）。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        #wrapper
        {
            width:240px;
            height:20px;
            border:1px solid black;
        }
        #first, #second
        {
            width:60px;
            height:60px;
            border:1px solid gray;
            margin-top:10px;
            margin-left:10px;
            margin-right:10px;
```

```
            background-color: #F4F6F4;
        }
        #first{float:left;}
        #second{float:right;}
    </style>
</head>
<body>
    <div id="wrapper">
        <div id="first">1</div>
        <div id="second">2</div>
    </div>
</body>
</html>
```

在浏览器预览效果如图9-15所示。

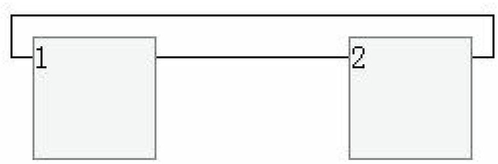


图9-15　不同方向的兄弟元素

2. 兄弟元素不是浮动元素

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        #wrapper
        {
            width:200px;
            height:150px;
            border:1px solid red;
        }
        /*第一个div设置浮动*/
        #first
        {
```

```
            width:80px;
            height:80px;
            border:1px solid gray;
            float:left;
            background-color:#F4F6F4;
        }
        /*第二、三个div没有设置浮动*/
        #second,#third
        {
            width:100px;
            height:30px;
            border:1px solid gray;
        }
    </style>
</head>
<body>
    <div id="wrapper">
        <div id="first">1</div>
        <div id="second">2</div>
        <div id="third">3</div>
    </div>
</body>
</html>
```

在浏览器预览效果如图9-16所示。

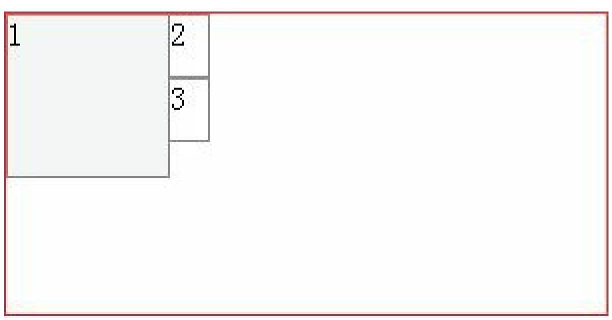


图9-16　兄弟元素不是浮动时效果

分析：

在这个例子中，第一个div设置了浮动，第二、三个div没有设置浮动。此时可以看出，第一个div脱离了文档流，并且覆盖了第二、三这

两个div。

当我们也为第二、三个div设置浮动（float:left）之后，在浏览器预览效果如图9-17所示。此时可以看出，第一个div不再覆盖第二、三个div。

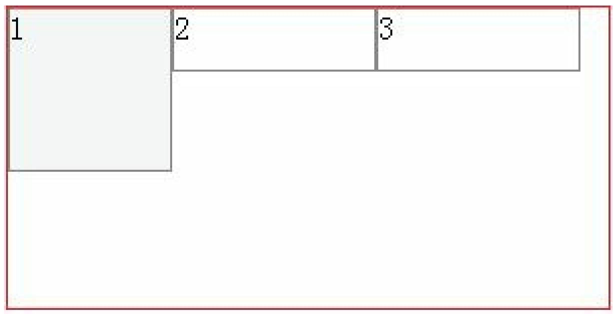


图9-17　兄弟元素是浮动时效果

从上面这个例子可以看出，浮动的影响很奇葩，对布局影响很大。因此在实际开发中我们使用了浮动之后，尽量清除浮动，不然会有预想不到的后果。

9.3.4　对子元素的影响

从“浮动元素对其父元素的影响”我们知道，如果父元素没有定义高度height，此时浮动元素会脱离父元素，造成父元素高度塌陷。但是当父元素同时也是一个浮动元素的时候，这个父元素会自适应地包含该子元素。

也就是说，如果一个元素是浮动元素（没有定义height），并且它的子元素也是浮动元素，则这个浮动元素会自适应地包含该子元素。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
```

```
    <style type="text/css">
        #wrapper
        {
            width:200px;
            border: 1px solid black;
        }
        #first,#second
        {
            width:80px;
            height:40px;
            border:1px solid red;
        }
        #first{float:left;}
        #second{float:right;}
    </style>
</head>
<body>
    <div id="wrapper">
        <div id="first"></div>
        <div id="second"></div>
    </div>
</body>
</html>
```

在浏览器预览效果如图9-18所示。

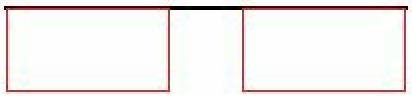


图9-18　父元素没有设置浮动

分析：

在这个例子中，如果我们给父元素定义“float:left”或者“float:right”，在浏览器预览效果如图9-19所示。

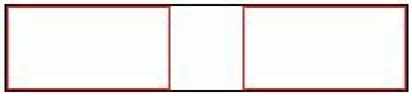


图9-19　父元素设置浮动

9.4 浮动的负作用

浮动可以让我们灵活地布局，但是也会带来一定的负作用。浮动带来的最常见的负作用有两个。

（1）父元素高度塌陷，从而导致边框不能撑开，背景色无法显示。

（2）页面布局错乱。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        #wrapper
        {
            width:200px;
            border:1px solid black;
            background-color:#51DCFF;
        }
        #first,#second
        {
            width:80px;
            height:40px;
            border:1px solid red;
        }
        #first{float:left;}
        #second{float:right;}
    </style>
</head>
<body>
    <div id="wrapper">
        <div id="first"></div>
        <div id="second"></div>
    </div>
```

```
</body>
</html>
```

在浏览器预览效果如图9-20所示。

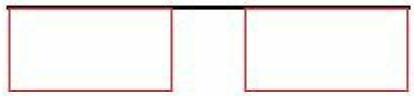


图9-20　浮动的负作用

分析：

在这个例子中，由于父元素没有定义高度，因此父元素无法把子元素包裹起来，就会造成父元素高度塌陷，从而导致父元素边框不能撑开并且背景色无法显示。当我们为父元素添加“overflow:hidden”来清除浮动之后，边框能够撑开了并且背景色也能显示出来，预览效果如图9-21所示。

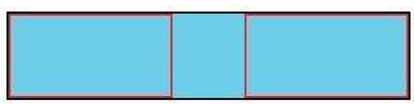


图9-21　清除浮动后效果

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        #wrapper
        {
            width:200px;
            height:150px;
            border:1px solid red;
```

```
        }
        /*第一个div设置浮动*/
        #first
        {
            width:80px;
            height:80px;
            border:1px solid gray;
            float:left;
            background-color:#F4F6F4;
        }
        /*第二、三个div没有设置浮动*/
        #second,#third
        {
            width:100px;
            height:30px;
            border:1px solid gray;
        }
    </style>
</head>
<body>
    <div id="wrapper">
        <div id="first">1</div>
        <div id="second">2</div>
        <div id="third">3</div>
    </div>
</body>
</html>
```

在浏览器预览效果如图9-22所示。

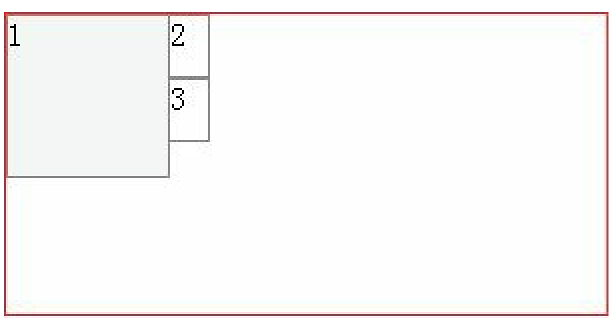


图9-22　浮动引起的布局错乱

分析：

在这个例子中，第一个div设置了浮动，第二、三个div没有设置浮动。此时可以看出，第一个div脱离了文档流，并且覆盖了第二、三这两个div，引起了布局的错乱。

给大家一个很实用的建议：在实际开发的过程中，如果我们因为写了某段CSS之后，突然发现页面布局错乱十分严重，此时我们首先应该想到的是浮动带来的负作用，然后认真检查一下是否没有清除浮动。

9.5　清除浮动

清除浮动，其实就是清除元素被定义浮动之后带来的脱离文档流的影响。我们知道，浮动可以使得元素移到左边或者右边，然后后面的文字或元素会环绕着这个浮动元素。如果我们不想浮动元素后面的元素环绕着它，而是希望后面的元素回归到正常文档流时候的布局，这个时候我们可以使用清除浮动来实现。

所谓的清除浮动，就是强制去除元素浮动之后带来的影响（这些影响在前面有过详细的介绍），然后使得其他元素都回归到正常文档流。

在CSS中，常见的清除浮动的方法有三种。

（1）clear:both。

（2）overflow:hidden。

（3）::after伪元素。

9.5.1　clear:both

在CSS中，我们可以使用clear属性来清除浮动。clear属性取值有三种：left、right和both。一般情况下，我们果断使用“clear:both”来清除所有浮动，非常省事。

这里注意一点，clear属性不是应用于浮动元素本身，而是应用于浮动元素后面的元素。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
```

```
    <style type="text/css">
        #wrapper
        {
            width:200px;
            border: 1px solid black;
        }
        #first,#second
        {
            width:80px;
            height:40px;
            border:1px solid red;
        }
        #first{float:left;}
        #second{float:right;}
        .clear
        {
              clear:both;     /*关键代码，清除浮动*/
        }
    </style>
</head>
<body>
<div id="main">
    <div id="wrapper">
        <div id="first"></div>
        <div id="second"></div>
        <div class="clear"></div>
    </div>
</div>
</body>
</html>
```

在浏览器预览效果如图9-23所示。

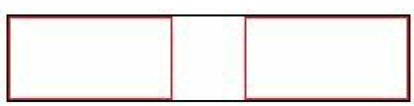


图9-23　使用“clear:both”清除浮动

分析：

为了清除元素的浮动，往往会添加一个div标签。很多时候这个div

标签仅仅是为了清除浮动而添加的，没有任何其他意义。这种是很多新手清除浮动的做法，但是这个方法破坏了HTML代码的语义，如果页面要清除多次浮动，这样就无缘无故添加了很多多余的div标签。

9.5.2　overflow:hidden

在CSS中，我们可以使用“overflow:hidden”来清除浮动。这里要注意一点，“overflow:hidden”应用于浮动元素的父元素，而不是当前的浮动元素。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        #wrapper
        {
            overflow:hidden;       /*关键代码，清除浮动*/
            width:200px;
            border:1px solid black;
        }
        #first,#second
        {
            width:80px;
            height:40px;
            border:1px solid red;
        }
        #first{float:left;}
        #second{float:right;}
    </style>
</head>
<body>
    <div id="main">
    <div id="wrapper">
        <div id="first"></div>
        <div id="second"></div>
    </div>
</div>
```

```
</body>
</html>
```

在浏览器预览效果如图9-24所示。

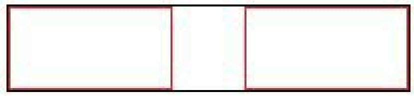


图9-24　使用“overflow:hidden”清除浮动

分析：

使用“overflow:hidden”清除浮动相对“clear:both”清除浮动来说，避免添加了多余的标签，并且不会破坏HTML的语义结构。不过“overflow:hidden”是个小炸弹，它会隐藏超出父元素的内容部分，有时候这并不是我们预期的效果。

9.5.3　::after伪元素

使用“clear:both”和“overflow:hidden”来清除浮动都有明显的弊端。在实际开发中，比较好的解决方案是使用“::after伪元素”结合“clear:both”来实现。

语法：

```
.clearfix{*zoom:1;}
.clearfix::after
{
    clear:both;
    content:"";
    display:block;
    height:0;
    visibility:hidden;
}
```

说明：

::after伪元素结合“clear:both”来清除浮动的方式，我们推荐用来定义成公共类（如类名为clearfix），然后进行全局引用，以便减少CSS代码。

其中::after是伪元素，在CSS3的动画效果中使用比较多。::before和::after都是常用的伪元素，建议大家自行了解一下，此处不展开介绍。“*zoom:1;”用于解决IE6、IE7浮动问题。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        .clearfix{*zoom:1;}
        .clearfix::after
        {
            clear:both;
            content:"";
            display:block;
            height:0;
            visibility:hidden;
        }
        #wrapper
        {
            width:200px;
            border:1px solid black;
        }
        #first,#second
        {
            width:80px;
            height:40px;
            border:1px solid red;
        }
        #first{float:left;}
        #second{float:right;}
```

```
    </style>
</head>
<body>
    <div id="wrapper" class="clearfix">
        <div id="first"></div>
        <div id="second"></div>
    </div>
</body>
</html>
```

在浏览器预览效果如图9-25所示。

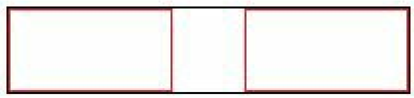


图9-25 “::after伪元素”清除浮动

分析：

“clear:both”清除浮动会增加多余标签，“overflow:hidden”清除浮动会使得超出父元素部分隐藏。但是使用::after伪元素来清除浮动，都不会有这些缺点。在实际开发中，我们也更倾向于使用这种方法。

浮动涉及的理论知识非常多，其中包括：块元素和行内元素、CSS盒子模型、脱离文档流、BFC、层叠上下文。建议大家一定要结合一下后面介绍的“BFC”和“层叠上下文”这两个，会对浮动有更深刻的理解。

第10章　定位布局

10.1　深入定位

浮动和定位是CSS两大布局方式。浮动布局比较灵活，但不容易控制。定位布局虽然缺乏灵活性，但是却可以让用户精准地定位页面中元素的位置。在CSS中，定位布局共有四种方式。

（1）固定定位（fixed）。

（2）相对定位（relative）。

（3）绝对定位（absolute）。

（4）静态定位（static）。

对于定位布局，我们需要注意以下几点。

（1）默认情况下，固定定位元素和绝对定位元素的位置是相对于浏览器而言，而相对定位元素的位置是相对原始位置而言。这里注意一个前提——默认情况下。

（2）position属性一般配合top、bottom、left和right来使用。只有元素定义position属性（除了static）之后，top、bottom、left和right才生效。

（3）top、bottom、left和right这四个属性不一定全部都用到。

（4）“position:absolute”会将元素转换为块元素。

10.1.1　子元素相对父元素定位

在入门阶段，我们都知道定位往往都是相对于浏览器或者原始位置而言的。但是在实际开发中，我们经常要实现子元素相对于父元素来定位，那该怎么办呢？

语法：

```
父元素{position:relative;}
子元素
{
    position:absolute;
    /*定义top、bottom、left和right*/
}
```

说明：

想要实现子元素相对父元素定位，我们都是给父元素定义“position:relative;”，然后给子元素定义“position:absolute;”，之后配合top、bottom、left和right来定位。这个技巧在实际开发中大量使用，它也是定位布局的精髓之一，大家一定要重点掌握。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        .father
        {
            position:relative;
            width:160px;
            height:30px;
```

举例：

```
            background-color:#00FFFF;
        }
        .son
        {
            position:absolute;
            bottom:-30px;
            right:65px;
            width:30px;
            height:30px;
            background-color:red;
        }
    </style>
</head>
<body>
    <div class="father">
        <div class="son"></div>
    </div>
</body>
</html>
```

在浏览器预览效果如图10-1所示。

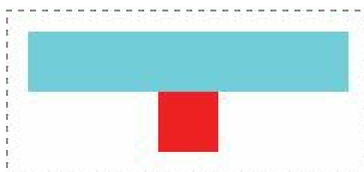


图10-1　子元素相对父元素定位（1）

分析：

使用上面这个技巧，我们可以把子元素随心所欲地相对父元素来定位。在这里我们注意一点，子元素没有使用“display:inline-block”，但是却可以定义width和height，为什么？其实，这是因为“position:absolute”会将元素转换为块元素。这是绝对定位非常重要的一个特点。 接下来，我们再来看一个复杂的例子。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        body{float:left;}
        .father
        {
            position:relative;
            width:120px;
            height:30px;
            background-color:#00FFFF;
        }
        .son
        {
            position:absolute;
            bottom:-20px;
            right:50px;
            width:20px;
            height:20px;
            background-color:red;
        }
    </style>
</head>
<body>
    <div class="father">
        <div class="son"></div>
    </div>
    <div class="father">
        <div class="son"></div>
    </div>
    <div class="father">
        <div class="son"></div>
    </div>
</body>
</html>
```

在浏览器预览效果如图10-2所示。

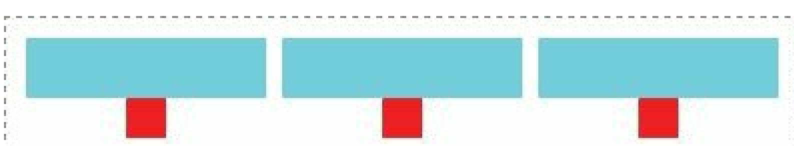


图10-2　子元素相对父元素定位（2）

分析：

子元素相对父元素定位用途非常多，例如二级导航、微调元素位置等。像绿叶学习官网导航中的三角形效果，其实也是使用子元素相对父元素定位来实现的，如图10-3所示。



图10-3　绿叶学习网中的绝对定位

10.1.2　子元素相对祖先元素定位

上面这两个例子都是子元素相对父元素来定位的，但是有些时候我们想要实现子元素相对祖先元素（例如祖父元素），那怎么办呢？

语法：

```
祖先元素{position:relative;}
子元素
{
    position:absolute;
    /*定义top、bottom、left和right*/
}
```

说明：

想要实现子元素相对祖先元素定位，我们都是给祖先元素定义“position:relative;”，然后给子元素定义“position:absolute;”，之后配合top、bottom、left和right来定位。这与子元素相对于父元素定位是一样的道理。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        .grandfather
        {
            position:relative;   /*设置相对定位*/
            width:200px;
            height:160px;
            background-color:blue;
        }
        .father
        {
            position:relative;   /*设置相对定位*/
            width:120px;
            height:30px;
            background-color:#00FFFF;
        }
        .son
        {
            position:absolute;  /*设置绝对定位*/
            bottom:-20px;
            right:50px;
            width:20px;
            height:20px;
            background-color:red;
        }
    </style>
</head>
<body>
    <div class="grandfather">
        <div class="father">
            <div class="son"></div>
        </div>
    </div>
</body>
</html>
```

在浏览器预览效果如图10-4所示。

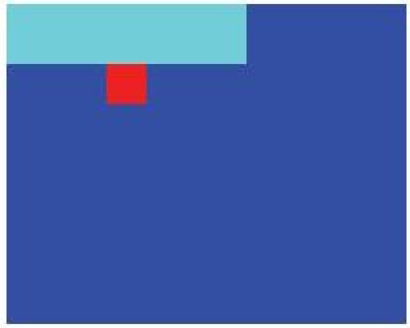


图10-4　实际效果图

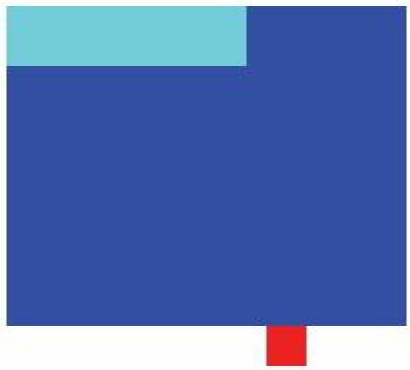


图10-5　预期效果图

分析：

小伙伴们可能就有疑问了：奇怪了，为什么这里的子元素不是相对祖父元素来进行定位的，而还是相对于父元素来进行定位呢？预期的效果应该像图10-5这样啊。

这是因为祖父元素定义了“position:relative;”的同时，父元素也定义了“position:relative;”，因此子元素依旧是相对于父元素来定位的。我们在CSS中删除父元素“position:relative;”这条属性之后，就可以发现子元素会相对于祖父元素来定位了。

绝对定位元素是相对于外层第一个设置了“position:relative;”“position:absolute;”或“position:fixed”的祖先元素来进行定位的。这个规律极其重要，大家请好好琢磨这句话。

此外，关于定位布局的基本语法不在本书详细展开，具体内容请查看《Web前端开发精品课HTML和CSS基础教程》相关章节。

10.2　z-index属性

虽然网页是平面的，但实际上网页是三维结构，除了x轴、y轴，它还有z轴。z轴往往都是用来设定层的先后顺序的。

在CSS中，我们可以使用z-index属性来控制z轴的大小，从而控制元素的堆叠顺序，如图10-6和图10-7所示。也就是说，我们可以使用z-index属性将一个元素放置于另外一个元素的下面。

默认情况下，元素的z-index属性处于不激活状态。也就是说默认情况下，设置元素的z-index属性无效。z-index属性只有在元素定义“position:relative”“position:absolute或者“position:fixed”时才会被激活。当然，对于“position:fixed”的z-index也没什么值得去管，直接忽略即可。

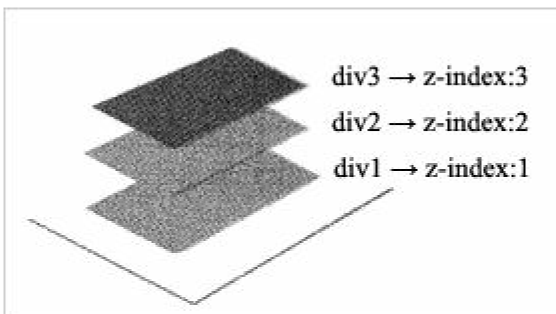


图10-6　z-index原理图（1）

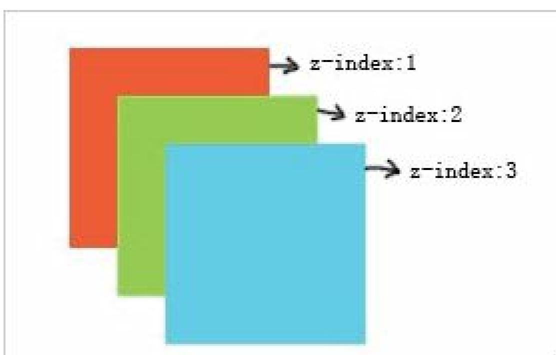


图10-7　z-index原理图（2）

语法：

```
z-index:取值;
```

说明：

z-index属性取值如表10-1所示。

表**10-1　z-index**取值

属　性　值	说　　明
auto	默认值，堆叠顺序与父元素相等
number	可以为负整数、0以及正整数
inherit	规定应该从父元素继承z-index属性的值

W3C标准中对CSS的z-index属性是这样定义的：“z-index属性设置元素的堆叠顺序，拥有更高堆叠顺序的元素总是会处于堆叠顺序较低的元素的前面。该属性设置一个定位元素沿z轴的位置，z轴定义为垂直延伸到显示区的轴。如果为正数，则离用户更近，为负数，则表示离用户更远。”

默认情况下，元素z-index属性值为auto。z-index值为正数的元素在z-index值为0的上面，z-index值为负数的元素在z-index值为0的下面。无论是正数还是负数，z-index值较大的元素会叠加在z-index值较小的元素之上。如果z-index值相同，则遵循“后来者居上”规则来叠加，如图10-8所示。

此外要记住一点，如果元素没有指定position属性值（除了

static），则z-index属性无效。

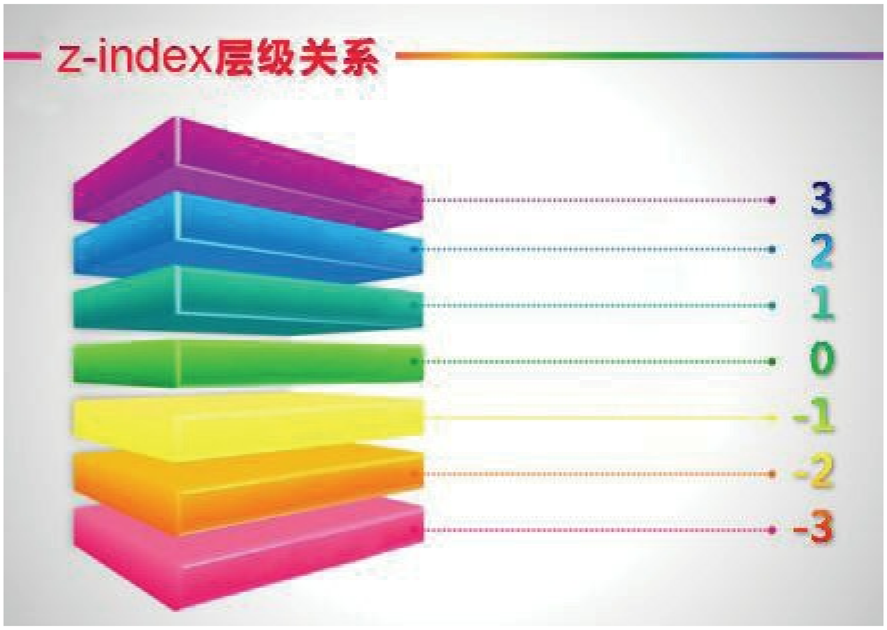


图10-8　z-index层级关系

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title></title>
    <style type="text/css">
        div
        {
            width:100px;
            height:100px;
            font-size:50px;
            position:absolute;
        }
        #A{background-color:red;top:10px;left:10px;}
        #B{background-color:orange;top:40px;left:40px;}
        #C{background-color:blue;top:70px;left:70px;}
    </style>
</head>
<body>
```

```
    <div id="A">A</div>
    <div id="B">B</div>
    <div id="C">C</div>
</body>
</html>
```

在浏览器预览效果如图10-9所示。

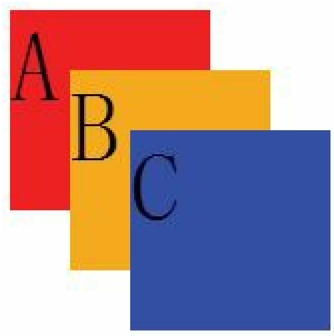


图10-9　没有设置z-index时效果

分析：

在这个例子中<div id="A">A</div>、<div id="B">B</div>和<div id="C">C</div>在HTML文档中从上到下显示，因此在浏览器看到的效果为：A容器在最下面，B在中间，C在最上面。假如给A容器加上“z-index:3”，给B容器加上“z-index:2”，给C容器加上“z-index:1”，在浏览器预览效果如图10-10所示。A、B、C三个div元素z-index属性值分别是3、2、1，根据z-index值大小来排列顺序。

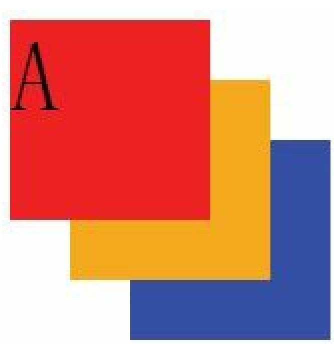


图10-10　设置z-index时效果

这一节只是简单介绍一下z-index属性的基本语法。实际上z-index属性涉及的东西很多，例如层叠上下文。对于层叠上下文，我们在后面章节会详细介绍。

第11章　CSS图形

11.1　CSS图形简介

在浏览网页的过程中，我们经常看到各种图形效果，如图11-1至图11-3所示。对于这些图形效果，小伙伴们更多时候想到的是用图片来实现。但是在前端开发中，为了网站的性能速度，我们都是秉着“少用图片”的原则。因为图片实现有两个很明显的缺点：一是图片大小比较大，数据传输量大；二是一张图片会引发一次HTTP请求。这两个方面都会影响页面加载速度，并且增加服务器负担。试想一下，作为用户，如果打开一个网页延迟时间过长，你是怎样的心情？

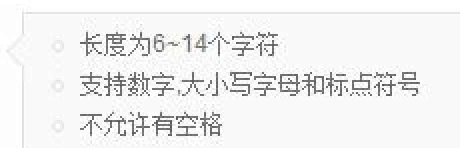


图11-1　网页中的三角形



图11-2　网页中的圆角

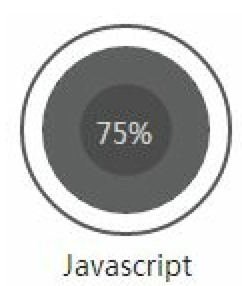


图11-3　网页中的圆

在实际开发中，对于这些图形效果，我们更倾向于使用CSS而非图片来实现。在这一章里，我们主要介绍以下三种图形的CSS实现方式。

（1）三角形。

（2）圆角与圆。

（3）椭圆。

以上三种图形的实现是在CSS中最常见的。除了这些图形，像多边形这种比较少见，因此在这一章就不展开介绍了。不过使用CSS可以实现各种多边形效果，例如梯形、五角星、钻石等，还是挺有趣的，我们可以自行搜索了解一下。

CSS实现的图形一般适合于展示，并不适合用于JavaScript动态操作。如果想要实现便于JavaScript操作的图形，大家可以去了解canvas或SVG。这两个可以实现各种酷炫的动态图形效果，例如粒子碰撞、动感圆圈（如图11-4所示）等。

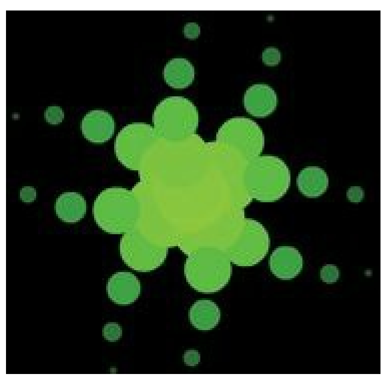


图11-4　canvas实现的动感圆圈

11.2　三角形

三角形在很多地方都能见到，如下拉菜单、表单注册、用户消息等。图11-5至图11-7这些三角形，都是使用CSS而并非图片来实现的。居然可以用CSS来实现三角形，同学们是不是觉得很神奇？当初我接触的时候，心情跟你们是一样的激动。

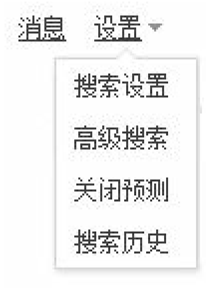


图11-5　下拉菜单中的三角形

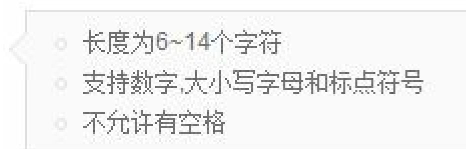


图11-6　表单注册中的三角形



图11-7　用户消息中的三角形

11.2.1　CSS实现三角形的原理

在CSS盒子模型中，当一个盒子的两条边在边角处相交时，浏览器就会在交点处按照某个角度（如果盒子为正方形，则为顺时针45°、135°、225°、315°）绘制一条接合线。我们先来看一个例子，更容易理解些。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        #box
        {
            width:30px;
            height:30px;
            border-width:20px;
            border-style:solid;
            border-color:red green blue orange;
        }
    </style>
</head>
<body>
    <div id="box"></div>
</body>
</html>
```

在浏览器预览效果如图11-8所示。

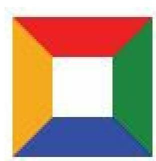


图11-8　width和height不为0的盒子效果

分析：

在这个例子中，我们为每一条边框定义不同的颜色（border-color），并且设置足够大的宽度（border-width），然后可以很明显地看出两条边相交时的效果。如果我们把盒子的宽度（width）和高度（height）都定义为0时，会得出图11-9所示的效果。



图11-9　width和height都为0的盒子效果

我们都知道，border-color属性包含四个值，分别对应“上、右、下、左”四条边颜色，呈顺时针排列。如果将“右、下、左”这三条边颜色改为transparent（透明）会发生什么呢（实际代码如下）？哇！这个时候呈现了一个指向下方的三角形，如图11-10所示。

```
<style type="text/css">
    #box
    {
        border-width:20px;
        border-style:solid;
        border-color:red transparent transparent transparent;
    }
</style>
```

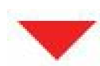


图11-10　“右、下、左”这三条边颜色改为transparent时的效果

当然我们也可以定义“下、左”两条边颜色为transparent，这会实现一个指向右上方的三角形，如图11-11所示。

```
<style type="text/css">
    #box
    {
        border-width:20px;
        border-style:solid;
        border-color:red red transparent transparent;
    }
</style>
```

从上面我们可以总结出使用CSS来实现三角形的原理：将一个元素的width和height定义为0，然后为它设置较粗的边框，并且将其中任意三条边框或者两条边的颜色定义为transparent。

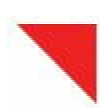


图11-11　将“下、左”这两条边的颜色改为transparent的效果

注意，上面例子中所有边框的border-width都是相同的，我们可以通过定义不同的border-width来改变三角形的形状，如图11-12所示。使用CSS实现三角形的原理都是相同的。我们可以思考一下以下这些三角形是如何实现的。

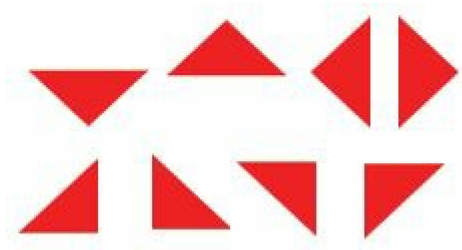


图11-12　CSS实现的三角形效果

11.2.2　带边框的三角形

在实际开发中，我们经常要实现如图11-13所示的“带边框的三角形”。由于三角形本身就是border，我们不可能通过给border添加border属性来实现。

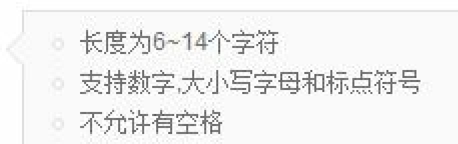


图11-13　带边框的三角形

对于这种带边框的三角形，我们一般使用两个三角形来实现。一个作为背景色（内层三角形），一个作为边框色（外层三角形），然后通过定位布局重叠在一起。注意，两个三角形定位要相差1像素。一般情况下，都是将内层三角形相对于外层三角形进行定位，偏移1像素。

在实现带边框的三角形原理中，有一个关于绝对定位的问题是一定不可忽视的：上、右、下、左四个方向三角形相对于父元素定位是不同的。必须把这个问题理解清楚，我们才能深刻理解带边框三角形的实现原理。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        /*外层三角形*/
        #triangle
        {
            position:relative; /*设置position:relative，使子元素可以相对父元素i
            width:0;
            height:0;
            border-width:30px; /*注意外层三角形高为30px*/
            border-style:solid;
            border-color:transparent transparent black transparent;
        }
        /*内层三角形*/
        #triangle div
        {
            position:absolute;
            top:1px;
            left:0;
            width:0;
            height:0;
            border-width:29px;/*注意内层三角形边高为29px*/
            border-style:solid;
            border-color:transparent transparent #BBFFEE transparent;
        }
    </style>
</head>
<body>
    <div id="triangle">
        <div></div>
    </div>
</body>
</html>
```

在浏览器预览效果如图11-14所示。

分析：

举例：

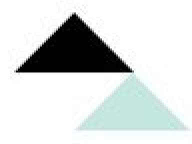


图11-14　带边框三角形实际效果

外层三角形高为30px，内层三角形高为29px。按道理说，如果内层三角形top定义为1px（向下移动1px），left定义为0时，预览效果应该如图11-15所示，但为什么跟预期效果不一样呢？

其实在CSS中，子元素的绝对定位是根据父元素的“内容边界（content）”进行定位的。也就是说，“内层三角形对应的盒子”的绝对定位是根据“外层三角形对应的盒子”的内容边界content来进行定位的，而不是根据我们肉眼所看到的三角形的边界来进行定位的，如图11-16所示。由于盒子width和height都是0，因此content是在盒子的中心（也就是中心点）。

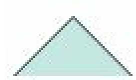


图11-15　带边框三角形预期效果

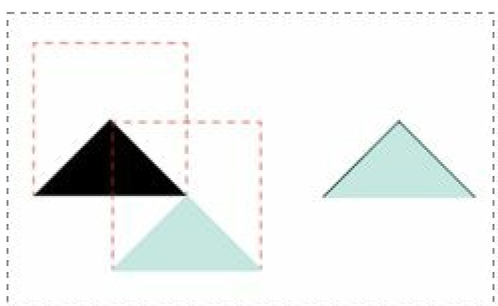


图11-16　带边框三角形分析图

如果想要实现图11-14的效果，top应该为-28px，left应该为-29px。对于上面提及的绝对定位的原理，我们多结合盒子模型来理解就很容易了。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        #wrapper
        {
            display:inline-block;
            position:relative;
            padding:20px 30px;
            margin-top:100px;
            border:1px solid gray;
            border-radius:10px;/*添加圆角效果*/
            font-size:14px;
            font-weight:bold;
            text-align:center;
            background-color:#BBFFEE;
        }
        /*外层三角形*/
        #triangle
        {
            position:absolute;
            top:-30px;
            /*left:50%和margin-left:-15px是为了实现三角形的水平居中*/
            left:50%;
            margin-left:-15px;
            width:0;
            height:0;
            border-width:15px;
            border-style:solid;
            border-color:transparent transparent black transparent;
        }
        /*内层三角形*/
        #triangle div
        {
            position:absolute;
            top:-13px;
            left:-14px;
            width:0;
            height:0;
            border-width:14px;
            border-style:solid;
            border-color:transparent transparent #BBFFEE transparent;
        }
    </style>
</head>
<body>
```

```
    <div id="wrapper">
        <div id="triangle"><div></div></div>
        欢迎来到绿叶学习网
    </div>
</body>
</html>
```

在浏览器预览效果如图11-17所示。

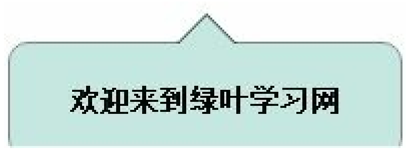


图11-17　带边框三角形应用

分析：

上面例子的CSS代码有点多，但是核心代码只有那么一点点。要想实现对话气泡效果，我们需要进行两次定位：一次是将外层三角形和内层三角形作为一个整体相对于容器进行定位，另外一次是将内层三角形相对于外层三角形进行定位。

一般情况下，外层三角形border-width比内层三角形borderwidth大1px。此外，外层三角形的left值一般都是其border-wdith的负数，top值一般都是其border-width的负数加1。例如在上面这个例子中，外层三角形border-wdith为14px，则left应该定义为-14px，top应该定义为-13px。

11.3　圆

我们在很多网站，包括绿叶学习网，都经常能看到各种圆角效果，如图11-18和图11-19所示。圆角效果往往更为美观大方，用户体验也更好。



图11-18　圆角按钮



图11-19　圆角图标

11.3.1　CSS实现圆角

在CSS2.1中，给元素添加圆角效果是一件很头疼的事情。老办法都是使用背景图片来实现，制作起来比较麻烦。CSS3中border-radius属性的出现，使得圆角效果的实现得到完美的解决。

1．border-radius属性

在CSS中，对于圆角效果我们都是使用CSS3的border-radius属性来实现。

语法：

```
border-radius:长度值;
```

说明：

长度值可以是px、百分比、em等。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>CSS3 border-radius属性</title>
    <style type="text/css">
        div
        {
            width:100px;
            height:50px;
            border:1px solid gray;
            border-radius:10px;
        }
    </style>
</head>
<body>
    <div></div>
</body>
</html>
```

在浏览器预览效果如图11-20所示。



图11-20　border-radius属性简单实例

分析：

“border-radius:10px;”指的是元素四个角的圆角半径都是10px。

2．border-radius属性值的四种写法

border-radius属性跟border、padding、margin等属性相似，其属性值也有四种写法。

（1）border-radius设置一个值。

例如“border-radius:10px;”表示四个角的圆角半径都是10px，如图11-21所示。

（2）border-radius设置两个值。

例如“border-radius:10px 20px;”表示左上角和右下角的圆角半径是10px，右上角和左下角的圆角半径都是20px，如图11-22所示。



图11-21　border-radius设置一个值



图11-22　border-radius设置两个值

（3）border-radius设置三个值。

例如“border-radius:10px 20px 30px;”表示左上角圆角半径是10px，左下角和右上角的圆角半径都是20px，右下角圆角半径是30px，如图11-13所示。

（4）border-radius设置四个值。

例如“border-radius:10px 20px 30px 40px;”表示左上角、右上角、右下角和左下角的圆角半径依次是10px、20px、30px、40px，如图11-14所示。



图11-23　border-radius设置三个值

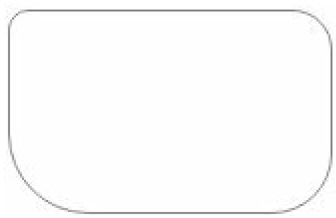


图11-24　border-radius设置四个值

这里的“左上角、右上角、右下角、左下角”，大家按照顺时针方向记忆就好了。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>CSS3 border-radius属性</title>
    <style type="text/css">
        div
        {
            width:200px;
            height:100px;
            border:1px solid red;
            border-radius:10px 20px 30px 40px;
            background-color:#FCE9B8;
```

```
        }
    </style>
</head>
<body>
    <div></div>
</body>
</html>
```

在浏览器预览效果如图11-25所示。



图11-25　border-radius实例（2）

分析：

大家可以自行在本地编辑器中为border-radius属性设置不同值，然后查看实际效果如何。

在实际开发中，border-radius属性一般都是设置一个值，使得四个圆角效果都一样。四个圆角都搞得不一样，有必要这么花哨么？当然有啊，像下面这种效果就是这么花哨。对于图11-26所示，关键是实现标签前面的圆形效果。



图11-26

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>CSS3 border-radius属性</title>
    <style type="text/css">
        div
        {
            width:50px;
            line-height:50px;
            border-radius:80% 90% 100% 20%;
            background-color:#E61588;
            font-size:30px;
            text-align:center;
            color:White;
        }
    </style>
</head>
<body>
    <div>6</div>
</body>
</html>
```

在浏览器预览效果如图11-27所示。



图11-27　border-radius实例（3）

11.3.2　CSS实现半圆和圆

1．半圆

半圆分为：上半圆、下半圆、左半圆、右半圆。我们只要学会制作某一个方向的半圆，其他方向的半圆都可以轻松实现，因为原理是一样的。

假如我们要制作上半圆，实现原理：把高度height设为宽度width的一半，并且左上角和右上角的圆角半径定义与元素的高度一致，而右下角和左下角的圆角半径定义为0。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>CSS3 border-radius属性</title>
    <style type="text/css">
        div
        {
            width:100px;
            height:50px;
            border:1px solid red;
            border-radius:100px 100px 0 0;
            background-color:#FCE9B8;
        }
    </style>
</head>
<body>
    <div></div>
</body>
</html>
```

在浏览器预览效果如图11-28所示。

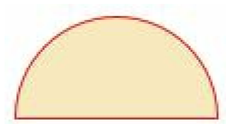


图11-28　使用border-radius实现半圆

分析：

在这个例子中，border-radius属性值等于圆角的半径。大家结合一下圆和矩形的数学知识，稍微想一想就知道上半圆如何实现了。

此外，请大家根据上面的原理自行测试一下下半圆、左半圆以及右半圆是如何实现的。

2. 圆

在CSS3中，圆的实现原理如下：元素的宽度和高度定义为相同值，然后四个角的圆角半径定义为宽度（或高度）的一半。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>CSS实现圆</title>
    <style type="text/css">
        div
        {
            width:100px;
            height:100px;
            border:1px solid red;
            border-radius:50px;
            background-color:#FCE9B8;
        }
    </style>
</head>
<body>
    <div></div>
</body>
</html>
```

在浏览器预览效果如下：

分析：

在这个例子中，width和height属性值相同，border-radius属性值为width（或height）的一半，然后就可以实现一个圆了，如图11-29所示。

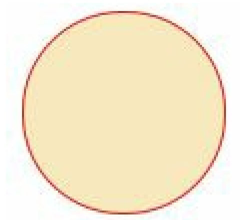


图11-29　使用border-radius实现圆

border-radius属性很强大，如图11-30所示的哆啦A梦其实就是用border-radius结合其他CSS属性来实现的，很神奇吧？我们可以自己尝试制作一下（本书附有源码）。



图11-30　使用border-radius实现哆啦A梦

11.3.3　border-radius派生子属性

border-radius属性可以分开，分别为四个角设置相应的圆角值，这四个角的属性如下。

（1）border-top-right-radius：右上角。

（2）border-bottom-right-radius：右下角。

（3）border-bottom-left-radius：左下角。

（4）border-top-left-radius：左上角。

11.4 椭圆

在CSS中，我们也是使用border-radius属性来实现椭圆的。

语法：

```
border-radius:x/y;
```

说明：

x表示圆角的水平半径，y表示圆角的垂直半径。从之前的学习中我们知道，border-radius属性取值可以是一个值，也可以是两个值。

当border-radius属性取值为一个值时，例如“border-radius:30px”，表示圆角水平半径和垂直半径为30px，也就是说“border-radius:30px”等价于“border-radius:30px/30px”，前者是后者的缩写，效果如图11-31所示。

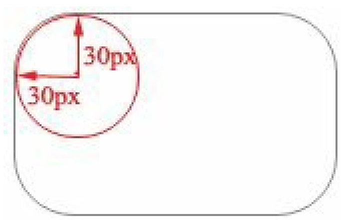


图11-31 “border-radius:30px”分析图

当border-radius属性取值为两个值时，例如“border-radius:20px/40px”，表示圆角的水平半径为20px，垂直半径为40px，效果如图11-32所示。

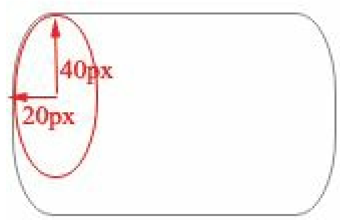


图11-32　“border-radius:20px/40px”分析图

如果想要实现椭圆，原理如下：元素的宽度和高度不相等，其中四个角的圆角水平半径定义为宽度的一半，垂直半径定义为高度的一半。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        div
        {
            width:160px;
            height:100px;
            border:1px solid gray;
            border-radius:80px/50px;
        }
    </style>
</head>
<body>
    <div></div>
</body>
</html>
```

在浏览器预览效果如图11-33所示。

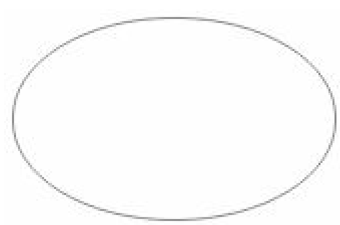


图11-33　使用border-radius实现椭圆

分析：

用CSS实现椭圆在实际开发中也比较常见。此外，我们可以尝试使用border-radius属性来实现如图11-34所示各种图形效果，以便加深理解。

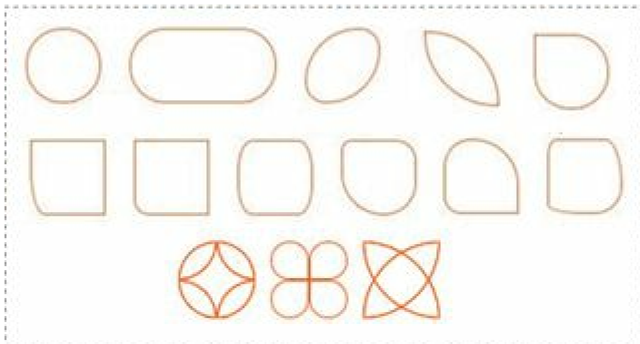


图11-34　使用border-radius实现各种图形效果

第12章　性能优化

12.1　CSS优化简介

编写高质量的CSS代码主要体现在2个方面：可读性和可维护性和高性能。对于一名前端工程师来说，如何平衡“追求高性能”和“可维护性”是一个很值得思考的问题。

对于流量比较少的普通网站来说，CSS本身的性能并不突出。因此，提高CSS代码的可读性和可维护性相对性能来说更重要一些。一般情况下，我们都是在确保CSS可读性和可维护性的前提下，再去考虑它的性能速度。但是对于大型网站，如淘宝，改善性能是非常具有实际意义的。

在这一章中，我们从以下八个方面来介绍一下CSS性能优化中的各种技巧。

（1）属性缩写。

（2）语法压缩。

（3）图片压缩。

（4）选择器优化。

（5）CSS模块化。

（6）压缩工具。

（7）CSS Sprite技术。

（8）性能评估。

CSS文件比较小，性能提高也是微乎其微。因此可能会有人说："我们如此细致地去优化CSS性能，意义并不大啊。"大多数情况下的确如此。但是当我们大规模地使用CSS，文件变得非常大，并且页面每天都会有几百万甚至上千万次的访问时，这种小小的性能提升就大不同了。对于一个流量比较少的小网站来说，CSS再怎么优化也提高不了多少性能。但是对于一个高流量的网站，如淘宝、百度等来说，CSS性能速度哪怕有一丁点提高也是非常有用的。

有研究表明，Amazon.com每增加10毫秒的页面加载时间会导致销售额下降1%，而谷歌搜索结果显示，加载时间每增加500毫秒将导致收入减少20%。我们可以看到性能的提高对于大型网站是多么重要了。哪怕所做的优化能够提高1毫秒的速度，也是相当有价值的。事实上，互联网大公司在性能考虑方面是非常细致而全面的。

此外，这些性能优化技能也是"真正前端工程师"和"野生网页设计师"重要分水岭之一。新手和高手写出来的代码，一眼就能够看出来。因此对于这一章的优化技巧，希望大家重点掌握。

12.2 属性缩写

在CSS中，很多属性是可以缩写的。属性缩写可以减少字符数，使得CSS代码量更少。对于属性缩写，我们主要从四个方面进行优化。

（1）盒模型缩写。

（2）背景缩写。

（3）字体缩写。

（4）颜色值缩写。

12.2.1 盒模型缩写

在CSS盒子模型中，有3个重要属性：border、padding和margin，如表12-1所示。

表**12-1** **CSS**盒子模型属性

属　　性	说　　明
border	边框
padding	内边距
margin	外边距

1．边框border

对于边框，我们需要定义3个方面：边框的宽度（border-width），边框的外观（border-style），边框的颜色（border-color）。

对于边框，有2种写法：完整形式和缩写形式。

（1）完整形式

```
border-width:1px;
border-style:solid;
border-color:Red;
```

（2）缩写形式

```
border:1px solid red;
```

在实际开发中，我们推荐使用border属性的缩写方式。“border:1px solid red”定义的是四条边样式，如果只想定义一条边样式，可以使用“border-top:1px solid red”这种方式。有时候，我们只想定义三条边的边框，可以采用以下方法。

```
border:1px solid red;
border-bottom:0;
```

或者：

```
border:1px solid red;
border-bottom:none;
```

2．内边距padding

padding写法有三种，分别如下

语法：

```
padding:长度值;
padding:长度值1长度值2;
padding:长度值1长度值2长度值3长度值4;
```

例如："padding:20px;"表示四个方向的内边距都是20px。

"padding:20px 40px;"表示padding-top和padding-bottom为20px，padding-right和padding-left为40px。"padding:20px 40px 60px 80px;"表示padding-top为20px，padding-right为40px，padding-bottom为60px，padding-left为80px。大家按照顺时针方向记忆就可以了，如图12-1所示。

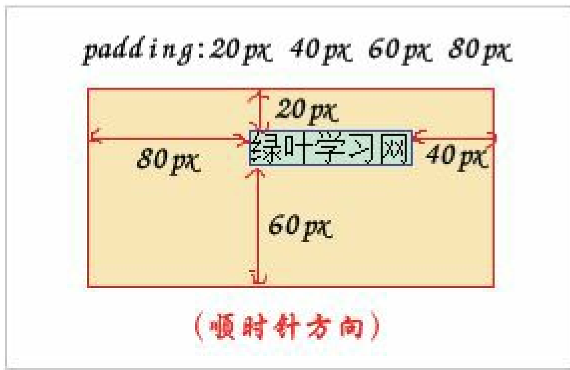


图12-1　padding简洁写法分析

3．外边距margin

margin写法跟padding写法相似，也有3种，分别如下。

语法：

```
margin:长度值;
margin:长度值1 长度值2;
margin:长度值1 长度值2 长度值3 长度值4;
```

例如："margin:20px;"表示四个方向的外边距都是20px。

"margin:20px 40px;"表示margin-top和margin-bottom为20px，margin-right和margin-left为40px。"margin:20px 40px 60px 80px;"表示margin-top为20px，margin-right为40px，margin-bottom为60px，margin-left为80px。大家按照顺时针方向记忆就可以了，如图12-2所示。

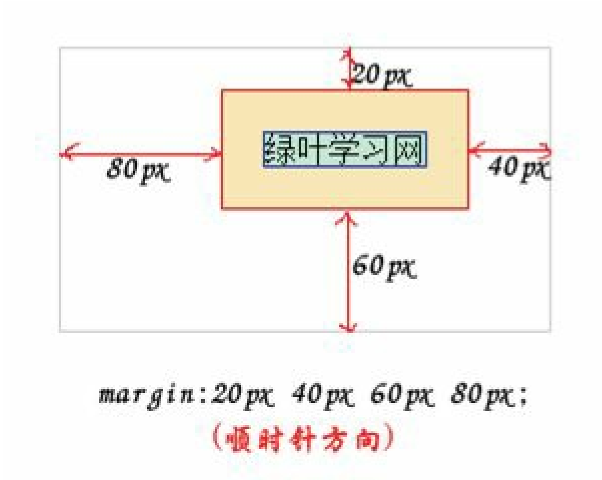


图12-2　margin简洁写法分析

对于border、padding和margin，要注意这几个属性的缩写方式是针对几个方向的，如果单独一个方向就没必要使用缩写形式，不然它连其他不需要的方向也定义了数值，会影响预期效果。

12.2.2　背景缩写

在CSS中，为元素定义背景，往往涉及表12-2所示属性。

表**12-2　CSS**背景属性

属　　性	说　　明
background-color	背景颜色
background-image	背景图片
background-repeat	背景重复
background-attachment	背景图片是固定还是滚动
background-position	背景图片的定位

对于背景，有两种写法：完整形式和简写形式。

1. 完整形式

```
background-image:url("images/flower.jpg");
background-repeat:no-repeat;
background-position:80px 40px;
```

2. 简写形式

```
background:url("images/flower.jpg") no-repeat 80px 40px;
```

在实际开发中，我们推荐使用简写形式。

12.2.3 字体缩写

在CSS中，常用到的字体以及文本属性如表12-3所示。

表**12-3 CSS**字体属性

属 性	说 明
font-family	字体类型
font-size	字体大小
font-weight	字体粗细
line-height	行高

对于背景，有两种写法：完整形式和简写形式。

1. 完整形式

```
font-family:"微软雅黑";
font-size:12px;
font-weight:bold
line-height:1.5em;
```

2. 简写形式

```
font:"微软雅黑" 12px/1.5em bold;
```

对于字体简写形式，需要注意以下两点。

（1）如果使用字体简写形式，我们至少要指定font-family和font-size属性，其他属性如果没有指定则将自动使用默认值。

（2）在简写形式中，font-size值和line-height值之间是需要加入斜杠“/”的。初学者要特别注意一下这种写法，在实际开发中会经常见到。

12.2.4 颜色值缩写

在CSS中，十六进制的颜色值是可以缩写的。如果每两位的值相同，可以缩写一半。例如“color:#000000”可以缩写为“color:#000”，“color:#336699”可以缩写为“color:#369”。当然在实际开发中，我们不采用缩写形式也没多大影响。之所以讲解，也是为了让大家知道有这种缩写形式，以免看不懂别人写的代码。

12.3　语法压缩

在CSS中，很多语法格式其实只是为了方便我们阅读，这些语法格式并不一定必须要使用。如果我们对于一些语法进行精简压缩，则可以减少CSS文件大小，从而减少页面数据传输量。

对于语法压缩，我们从以下7个方面进行优化。

（1）空白符。

（2）结尾分号。

（3）url()的引号。

（4）属性值为“0”。

（5）属性值为“以0开头的小数”。

（6）合并相同的定义。

（7）利用继承进行合并。

12.3.1　空白符

一般情况下，多个属性值之间才必须要用空格。在CSS中，空格和换行往往都是为了方便代码的阅读。

代码一（纵向书写）：

```
#wrapper
{
    padding:10px;
    border:1px solid gray;
}
```

```
#content
{
    font-size:14px;
    text-indent:2em;
}
```

代码二（横向书写）：

```
#wrapper{padding:10px;border:1px solid gray;}#content{font-size:14px;text-i
```

对于浏览器来说，上面两种书写方式是完全等价的。其中纵向书写代码方便阅读，横向书写则可以节省不少字符数。但是在实际开发中，我们建议使用纵向书写，不建议使用横向书写。然后等到整站发布的时候，再使用工具压缩成横向书写方式。当然，如果在实际开发中，某一个CSS规则只有一两个属性的时候，使用横向书写更为妥当。

12.3.2 结尾分号

在CSS中，每一个选择器的样式都是用大括号{}括起来。实际上，最后一个属性之后的结尾分号是不必要的。省略之后，对代码没有任何影响。

代码一：

```
#wrapper
{
    padding:10px;
    border:1px solid gray;
}
```

代码二：

```
#wrapper
{
    padding:10px;
    border:1px solid gray
}
```

对于浏览器来说，上面两段代码是等价的。省略最后一个分号可以让每一个规则减少一个字符。

12.3.3 url的引号

在CSS中，像background-image、cursor等属性url()中的路径不需要添加引号。

代码一：

```
h1
{
    background-image:url("logo.jpg");
    cursor:url("default.cur"),default;
}
```

代码二：

```
h1
{
    background-image:url(logo.jpg);
    cursor:url(default.cur),default;
}
```

对于浏览器来说，上面两段代码是等价的。省略引号可以让规则减少两个字符。

12.3.4 属性值为0

在CSS中，如果某一个属性取值为0，则这个属性值不需要添加单位。

代码一：

```
.test
{
    font-size:0px;
    padding:0em;
}
```

代码二：

```
.test
{
    font-size:0;
    padding:0;
}
```

对于浏览器来说，上面两段代码是等价的。

12.3.5　属性值为以0开头的小数

在CSS中，当一个属性的属性值是以0开头的小数时，我们可以把开头的0去掉。

代码一：

```
.test
{
    font-size:0.5em;
```

```
}
```

代码二：

```
.test
{
    font-size:.5em;
}
```

对于浏览器来说，上面两段代码是等价的。但是在实际开发中，不建议使用第二种方式，因为这种方式可读性比较差。不过，在整站发布之前，我们可以使用压缩工具来执行这个去掉0的操作。

12.3.6 合并相同的定义

在CSS中，很多时候定义的规则会有相同的部分，我们可以使用群组选择器来合并这些相同的样式，从而达到代码重用和精简代码的目的。

代码一：

```
.article
{
    font-size:14px;
    line-height:14px;
    text-indent:2em;
    background-color:#F1F1F1;
    border:1px solid silver;
}
.column
{
    font-size:14px;
    line-height:14px;
    text-indent:2em;
    background-color:orange;
```

```
    border:1px solid silver;
}
```

代码二：

```
.article,.column
{
    font-size:14px;
    line-height:14px;
    text-indent:2em;
    border:1px solid silver;
}
.article{background-color:#F1F1F1;}
.column{background-color:orange;}
```

代码二中使用群组选择器合并了两个不同规则之间的相同部分。这样一来，以后只要修改群组选择器中的CSS样式，就可以同时修改两个class的样式，非常利于代码的维护。

12.3.7　利用继承进行合并

在CSS中，很多属性是可以继承的。

（1）文本相关属性：font-family、font-size、font-style、font-weight、font、line-height、text-align、text-indent、word-spacing。

（2）列表相关属性：list-style-image、list-style-position、list-style-type、list-style。

（3）颜色相关属性：color。

如果父元素的多个子元素都定义了相同的可继承属性，我们可以把这些相同的属性定义在父元素上，从而精简代码。

代码一：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        #content
        {
            font-size:14px;
            font-weight:bold;
            line-height:14px;
            color:Red;
            background-color:Orange;
        }
        #sidebar
        {
            font-size:14px;
            font-weight:bold;
            line-height:14px;
            color:Red;
            background-color:#F1F1F1;
        }
    </style>
</head>
<body>
    <div id="wrapper">
        <div id="content"></div>
        <div id="sidebar"></div>
    </div>
</body>
</html>
```

代码二：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        #wrapper
        {
            font-size:12px;
            font-weight:bold;
```

```
            line-height:14px;
            color:Red;
        }
        #content{background-color:Orange;}
        #sidebar{background-color:#F1F1F1;}
    </style>
</head>
<body>
    <div id="wrapper">
        <div id="content"></div>
        <div id="sidebar"></div>
    </div>
</body>
</html>
```

代码二利用CSS属性的继承性，避免了代码的重复，精简了代码。

12.4　压缩工具

在前几节中我们学习了各种CSS的压缩技巧，这些对于我们深入了解CSS性能优化是很有帮助的。我们都知道CSS文件分为“开发版”和“发布版”。发布版是将开发版进行合并和压缩之后，在网站运行时使用的。

那么问题来了，在网站发布的时候如果我们要压缩CSS文件，是不是要逐项手动删除空白符、删除结尾分号、删除属性值为0的单位呢？说实话，如果现实是如此残酷的话，“程序猿”这种生物估计离绝种不远了。为了保护“程序猿”这种宝贵生物，前端界为我们提供了一大生存利器——CSS压缩工具！

CSS压缩工具有在线版、本地版和编辑器插件三种，这里推荐使用在线版的两个工具：CSS Compressor和YUI Compressor。

CSS Compressor网址：http://www.csscompressor.com

YUI Compressor网址：http://tool.oschina.net/jscompress

拿YUI Compressor来说，YUI Compressor会自动对CSS文件执行如下操作。

（1）删除所有注释。

（2）删除无用的空白符。

（3）删除结尾分号。

（4）删除属性值为0的单位。

（5）删除属性值以0开头的小数前的0。

（6）将相似属性合并，例如margin、padding、background等。

（7）将RGB颜色转换为十六进制颜色。

……

从上面可以看出，YUI Compressor真的为我们提供了一条龙服务呐。只需要轻轻一键压缩，你就可以高枕无忧了。想要了解YUI Compressor更多信息，大家自行搜索一下。

很多人会有一个疑问：是不是有了CSS压缩工具，平常我们就不需要注意代码的书写了呢？其实在实际开发中，还是建议大家养成良好的优化习惯，然后再使用压缩工具进行辅助。

12.5　图片压缩

随着Web页面设计的发展，越来越多的图片应用到了页面中，这也使得图片的加载和展示成为了Web前端比较突出的性能关注点。在很多门户网站中，图片的请求数往往接近总请求数一半以上。

12.5.1　JPEG、PNG和GIF

在实际开发中，JPEG、PNG和GIF是最常见的图片格式。深入理解这三种图片格式适合在哪种情况下使用，并且如何减少图片大小非常重要。

（1）JPEG可以很好地处理大面积色调的图像，适合存储颜色丰富的复杂图像，如照片、高清图片等。此外，JPEG不支持透明。

（2）PNG是一种无损格式，可以无损压缩以保证页面打开速度。此外，PNG支持透明。

（3）GIF格式图像效果较差，但是可以制作动画。

也就是说，如果想要展示色彩丰富而高品质图片，可以使用JPEG格式；如果是一般图片，为了减少体积，可以使用PNG格式；如果是动画图片，可以使用GIF格式。

关于JPEG、PNG和GIF更为详细的介绍，请参考本书的姊妹篇《Web前端开发精品课HTML和CSS基础教程》。

12.5.2　图片压缩

图片的传输量在一个页面的传输量中往往占的比重很高，因此对图片大小的压缩尤为重要。图片压缩工具很多，不过以在线工具居多。下面推荐两款比较好的在线工具。

（1）JPEG：推荐JPEGmini，网址：http://www.jpegmini.com

（2）PNG：推荐TinyPNG，网址：https://tinypng.org。这一款在线工具甚至比Photoshop的压缩率还要高，强烈推荐。

以上都是很好的在线工具，如果你想要使用本地软件，推荐ImageOptim这一款软件。ImageOptim是一款非常强大的工具，可以压缩JPEG、PNG以及GIF等各种格式的图片，可以将图片大小缩减30%～35%。此外还支持拖放功能，操作也非常简单。

ImageOptim下载地址：https://imageoptim.com/。

12.6　高性能的选择器

选择器是CSS中最常见的东西。但是很多人却不知道，不同的选择器其实性能是不一样的。了解选择器在浏览器中的解析原理以及不同选择器的解析速度，能够让我们的CSS性能速度锦上添花。

12.6.1　选择器在浏览器的解析原理

```
#column .content div{color:Red;}
```

一般情况下，我们都是从左到右来阅读代码的，因此对于上面这行代码，我们也会习惯性地以为浏览器是从左到右进行解析的：首先找到id为column的元素，然后再查找该元素下面class为content的元素，最后在已经匹配的元素下查找所有的div元素。

但是事实上却恰恰相反，浏览器对选择器规则是从右到左进行解析的：首先查找所有的div元素，然后再查找该div元素是否存在具有content类的父元素，最后在已经匹配的父元素中继续向上查找祖先元素是否含有id为column的元素。

当然，如果浏览器是从左到右解析选择器规则的话，上面这个选择器效率很高。但事实上浏览器是从右到左进行解析的，这种看似十分高效的选择器匹配开销是很高的，因为浏览器必须首先遍历页面所有的div元素，然后才确定其父元素的class是否为content。

看到这里，小伙伴们都会大吃一惊：“我out了！”

12.6.2　不同选择器的解析速度

举例：

```
#column .content div{……}
```

```
#column .test{……}
#test{……}
```

问大家一个问题，这3种选择器选中的都是同一个元素，那么性能最高的是哪个呢？实际上，第三个性能最好，第一个性能最差。由于第三个选择器直接使用id选择器，而id在整个页面具有唯一性，因此可以快速定位。第一个选择器需要先匹配所有的div元素，对于一个页面来说，这是个不小的匹配量。

浏览器解析选择器的原则是从右到左的，因此我们书写的最右边的选择器，被称为关键选择器。这个关键选择器，对于执行效率有决定性的影响。谷歌资深Web开发工程师Steve Souders对CSS选择器的匹配效率从高到低做了一个排序。

（1）id选择器。

（2）class选择器。

（3）元素选择器。

（4）相邻选择器。

（5）子选择器。

（6）后代选择器。

（7）通配符选择器。

（8）属性选择器。

（9）伪类选择器。

根据以上“选择器在浏览器的解析原理”以及“各种选择器的匹配效率”，如果我们想要更好地使用高性能的选择器，需要注意以下四个技巧。

1. 不要使用通配符

在选择器中，通配符一般用于选取页面中所有元素。例如“{}”表示

选取页面所有元素，“#wrapper *{}”表示选取id为wrapper元素下面的所有子元素。

通配符的匹配量非常大，一般情况下不建议使用。当然，从上面的CSS选择器匹配效率排序也可以看出来通配符的效率非常低。

2. 不要在**id**选择器以及**class**选择器前添加元素名

元素的id在一个页面中具有唯一性，因此在id选择器前添加元素名是多余的，同时也增加了匹配量。

元素的class不具有唯一性，如果在class选择器前添加元素名，则表示选择某一个class的某一种元素。除非是迫不得已的情况，否则尽量不要使用“class选择器前添加元素名”这种方式。

举例：

```
/*多余的写法*/
div#wrapper{font-size:12px;}
/*正确的写法*/
#wrapper{font-size:12px;}
```

3. 选择器最好不要超过三层，位置靠右的选择条件尽可能精确

选择器的层级越多，浏览器解析时匹配的次数就越多，因而速度就越慢。因此在定义选择器时，我们要尽量让选择器的层级少一些，最好不要超过三层。此外，根据选择器在浏览器中从右到左的解析原理可知，位置靠右的选择条件越精确，匹配量就越少，速度就越快。

4. 避免使用后代选择器，尽量少用子选择器

后代选择器匹配量比较大，应该避免使用。如果非要用的话，建议使用子选择器代替。但是子选择器匹配量也不小，如果有其他选择器如id选择器或class选择器等代替，也尽量少用子选择器。不过我们要注意

一下，尽量少用不等于不用，不要为了减少子选择器的使用而增加过多的id和class，以致id和class泛滥成灾。

【疑问】

现在浏览器解析速度那么快，为什么还要纠结选择器那一点点的性能提升呢？

之前已经明确说过了，对于小项目来说，这的确没多大影响。但是对于一个大型项目，特别是访问量每天达几百万次的网站来说，哪怕是一点点的性能提升也是非常重要的。当然，就算是做小项目，写一手优雅的代码是高手们的一种良好习惯。因为高手也讲究“诗意的栖居”。

当然对于小项目来说，我们还是在确保CSS的可读性和可维护性良好的前提下，再去考虑高性能的选择器。

第13章　CSS技巧

13.1　水平居中

实现文字、图片以及元素等的居中是CSS开发中必须掌握的技巧之一。在CSS中，居中包括两个方面：水平居中和垂直居中。当然居中对象也包括两个：文字和元素。这一节，我们先来给大家介绍一下CSS水平居中的技巧。

13.1.1　文字的水平居中

如果想要实现单行文字的水平居中，我们使用text-align属性就能轻松实现。多行文字的水平居中较少，在此可以忽略。

语法：

```
text-align:center;
```

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        div
```

```
        {
            display:inline-block;
            width:500px;
            height:50px;
            line-height:50px;
            text-align:center;   /*实现div元素内文字水平居中*/
            border:1px dashed gray;
        }
    </style>
</head>
<body>
    <div>CSS实现单行文字水平居中：text-align:center;</div>
</body>
</html>
```

在浏览器预览效果如图13-1所示。

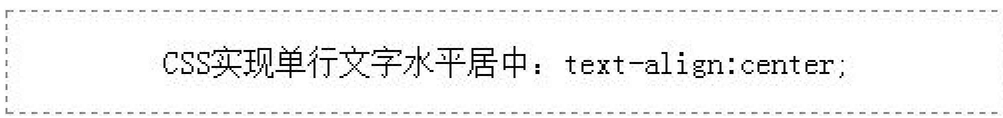


图13-1　文字的水平居中

13.1.2　元素的水平居中

1．块元素（block）

在浏览网页时，我们经常看到很多网站的布局，都是将内容整体进行水平居中。其实这是在最外层使用div来包裹，然后对div元素来实现水平居中。从本质上来说，div元素就是块元素。

在CSS中，对于块元素，如果给定了宽度，直接定义margin-left和margin-right都为auto就能实现水平居中。

语法：

```
margin:0 auto;
```

说明：

“margin:0 auto;”等价于“margin:0 auto 0 auto;”，也就是说真正起作用的是“margin-left:auto;”和“margin-right:auto;”，大家要理解这一点。从上面我们知道，如果想要在居中的同时加一个上外边距，可以写成“margin:20px auto 0 auto;”。如果想要在居中的同时加一个下外边距，可以写成“margin:0 auto 20px auto;”……也就是说，只要我们保证margin-left和margin-right都为auto，就能保证块元素的水平居中。

对于块元素来说，不管父元素的宽度如何，只要给块元素指定宽度，这个方法就有效。如果没有给块元素指定宽度，则块元素默认会占满允许的最大宽度，此时这个方法是没有效的。因此想要使用“margin:0 auto”来实现块元素的水平居中，就一定要指定块元素的宽度。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        div
        {
            margin:0 auto;
            width:80%;
            height:100px;
            border:1px solid gray;
        }
    </style>
</head>
<body>
    <div></div>
</body>
</html>
```

在浏览器预览效果如图13-2所示。

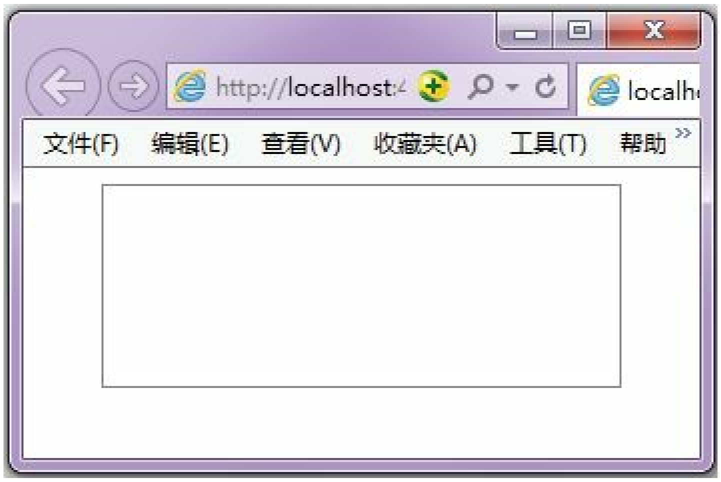


图13-2　块元素的水平居中

分析：

在这个例子中，需要水平居中的元素为div，div元素的父元素为body。这些逻辑大家要搞清楚。这个技巧在实际开发中经常用到，小伙伴们得认真掌握。

2. 行内元素（**inline**）以及复合行内元素（**inline-***）

对于行内元素（inline）以及复合行内元素（inline-*），我们可以使用“text-align:center”来实现水平居中。也就是说，“text-align:center”不仅可以用于文字，也可以用于行内元素以及复合行内元素。

复合行内元素包括inline-block、inline-table以及inline-flex之类的元素。

语法：

```
text-align:center;
```

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        div{text-align:center;}
    </style>
</head>
<body>
    <div><strong>strong元素</strong></div>
    <div><span>span元素</span></div>
    <div><a href="http://www.lvyestudy.com">a元素</a></div>
</body>
</html>
```

在浏览器预览效果如图13-3所示。



图13-3 inline元素的水平居中

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
```

```
    <title></title>
    <style type="text/css">
        body{text-align:center;}
        div
        {
            display:inline-block;
            width:100px;
            height:100px;
            border:1px solid gray;
        }
    </style>
</head>
<body>
    <div></div>
</body>
</html>
```

在浏览器预览效果如图13-4所示。

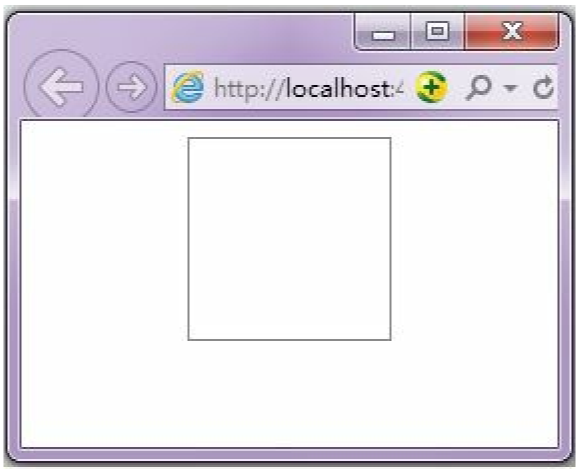


图13-4　inline-块元素的水平居中

分析:

此外我们要清楚一点,图片img也是inline-block元素。

13.2　垂直居中

对于垂直居中，我们也从“文字”和“元素”两个方面来给大家介绍一下，以便有一个清晰的学习思路。

13.2.1　文字的垂直居中

1. 单行文字

对于单行文字来说，我们定义line-height和height这两个属性的值相等就可以实现垂直居中。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        div
        {
            height:100px;
            line-height:100px;
            border:1px solid gray;
        }
    </style>
</head>
<body>
    <div>《Web前端开发精品课》</div>
</body>
</html>
```

在浏览器预览效果如图13-5所示。



图13-5　单行文字的垂直居中

分析：

为什么定义height和line-height这两个属性值相等，就可以实现单行文字的垂直居中呢？对于这一点，我们在第4章“深入line-height”这一节已经给大家详细介绍过了。

2．多行文字

如果父元素高度固定，文字可能是两行或者更多行，如何实现多行文字的垂直居中呢？

语法：

```
父元素
{
        display:table-cell;
        vertical-align:middle;
}
span{display:inline-block;}
```

说明：

实现的关键是，用一个span标签把所有文字包含起来，然后定义span为inline-block元素，之后使用inline-块元素垂直居中的方式来处理即可。对于inline-块元素垂直居中的实现方式，我们在下面会讲解。大家可以对比理解一下。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        p
        {
            display:table-cell;
            vertical-align:middle;
            width:400px;
            height:120px;
            border:1px dashed gray;
        }
        span{display:inline-block;}
    </style>
</head>
<body>
    <p>
        <span>《web前端开发精品课·HTML和CSS基础教程》<br/>
        《web前端开发精品课·HTML和CSS进阶教程》<br/>
        《web前端开发精品课·CSS3教程》</span>
    </p>
</body>
</html>
```

在浏览器预览效果如图13-6所示。



图13-6　多行文字的垂直居中

13.2.2　元素的垂直居中

1．块元素（block）

块元素的垂直居中一直很麻烦，对于高度已知的块元素，我们可以使用万能的position方法来实现。

使用position方法，父元素和子元素都必须定义宽度和高度，然后给父元素写上“position: relative”，这样做是为了给子元素添加“position:absolute”的时候不会被定位到外太空去。接下来给子元素添加如下属性：

```
position:absolute;
top:50%;
left:50%;
```

之后再添加如下属性：

```
margin-top:"height值一半的负值";
margin-top:"width值一半的负值".
```

语法：

```
父元素
{
    position:relative;
}
子元素
```

```
{
    position:absolute;
    top:50%;
    left:50%;
    margin-top:"height值一半的负值";
    margin-left:"width值一半的负值";
}
```

说明：

position这种方法是万能的，也就是不仅可以用于块元素，还可以用于inline、inline-block元素。对于margin-top和margin-left为什么要这样定义，大家自己画个草稿就能理解了。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title></title>
    <style type="text/css">
        #father
        {
            position:relative;
            width:200px;
            height:160px;
            border:1px solid gray;
        }
        #son
        {
            position:absolute;
            top:50%;
            left:50%;
            margin-top:-30px;
            margin-left:-50px;
            width:100px;
            height:60px;
            background-color:Red;
```

```
        }
    </style>
</head>
<body>
    <div id="father">
        <div id="son"></div>
    </div>
</body>
</html>
```

在浏览器预览效果如图13-7所示。

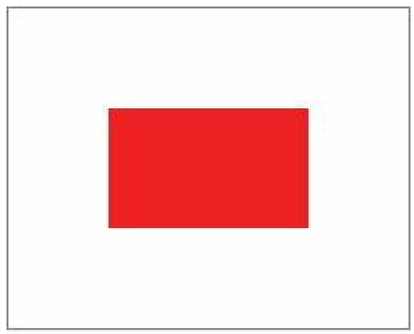


图13-7　块元素的垂直居中

分析：

对于position方法，我们需要注意以下两点。

（1）position方法可以用于所有元素，包括inline、inline-block、block元素。

（2）position方法可以实现水平和垂直两个方向同时居中，如果单独想要水平居中，把top和margin-top这两个属性去掉即可。如果想要单独实现垂直居中，把left和margin-left这两个属性去掉即可。

2．行内块元素（inline-block）

对于行内块元素的垂直居中，我们可以使用“display:table-cell”结合“vertical-middle”来实现。结合之前“display:table-cell”这一节介绍的内

容，我们很容易理解其原理。

语法：

```
父元素
{
        display:table-cell;
        vertical-align:middle;
}
子元素{vertical-align:middle;}
```

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title></title>
    <style type="text/css">
        div
        {
            display:table-cell;
            vertical-align:middle;
            width:240px;
            height:160px;
            border:1px solid gray;
        }
        img{vertical-align:middle;}
    </style>
</head>
<body>
    <div>
        <img src="images/haizei2.png" alt=""/>
    </div>
</body>
</html>
```

在浏览器预览效果如图13-8所示。



图13-8　inline-块元素的垂直居中

在实际开发中，对于水平居中和垂直居中的实现，除了这两节介绍这些方法，我们还可以考虑使用padding和margin来实现。

13.3　CSS Sprite

在浏览网页的过程中，我们经常可以看到很多网站都会使用各种图标、LOGO等，如图13-9和图13-10所示。这些图标很多时候是使用背景图片来实现的。有人会想，一个图标就使用一张背景图片来实现，这样岂不是很影响网站的性能？确实，如果一个图标就使用一张背景图片，那么每一个图标都会引发一次HTTP请求，极大地影响了网站的性能，并且难以维护。

一个好的解决办法就是把这些图标全部放到一张背景图片里面去，这样就只会引发一次HTTP请求了。如果把所有图标放到一张背景图片里面，那该怎么把图标拿出来使用呢？这个我们可以使用CSS中的background-position进行背景定位来取出相应的图标。上面这些，其实就是我们常说的CSS Sprite技术。



图13-9　淘宝中的小图标效果



图13-10　CSS Sprite图（雪碧图）

CSS Sprite，又称为“CSS精灵”或者“CSS雪碧图”。它将零散的小背景图合并成一张大的背景图，然后利用background-position属性进行背景定位从而显示相应的小背景图。其中，合并之后的大背景图，我们又常称为“雪碧图”。CSS Sprite听起来很高深，但是往往很深奥的名词都是拿来吓唬人的。

想要使用CSS Sprite，只需要简单的两步即可。

（1）使用Photoshop或者其他工具将小背景图合并成为一张大背景图，其中每一张小背景图要精确调整。

（2）使用background-image属性引入大背景图，并且结合background-position属性定位取出相应的图标。

对于CSS Sprite的使用，我们推荐两款非常不错的工具：CSS Sprite Generator和Sprite Cow。其中CSS Sprite Generator是一款在线工具，我们可以上传一个包含多个小背景图的压缩包，然后工具会自动生成大背景图（雪碧图）。此外，这款工具还可以自定义小背景图的位置、透明度以及背景色等。Sprite Cow可以用于自动生成“雪碧图”中某一个小背景图的CSS代码，这样我们就不需要那么麻烦地一个个地去取小背景图的具体位置像素了。

CSS Sprite Generator官方网址：http://css.spritegen.com/

Sprite Cow官方网址：http://www.spritecow.com

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        table{border-collapse:collapse;}
        table,tr,th,td{border:1px solid silver;}
        caption{font-weight:bold;margin-bottom:5px;}
        th
        {
            padding:3px;
            width:80px;
            font:微软雅黑 normal 14px/20px;
            font-weight:normal;
            background-color:#F1F1F1;
        }
        .chrome,.firefox,.ie,.opera,.safari
        {
            background: url(images/sprite.png) no-repeat;
            height:30px;
            padding-left:30px;
        }
        .chrome { background-position: -0px -0px; }
        .firefox { background-position: -0px -30px; }
        .ie { background-position: -0px -60px;}
        .opera { background-position: -0px -90px;}
        .safari { background-position: -0px -120px; }
    </style>
</head>
<body>
    <table>
        <caption>border-radius的浏览器支持情况</caption>
        <thead>
            <tr>
                <th>Chrome</th>
                <th>Firefox</th>
```

```
                <th>IE</th>
                <th>Opera</th>
                <th>Safari</th>
            </tr>
        </thead>
        <tbody>
            <tr>
                <td class="chrome"></td>
                <td class="firefox"></td>
                <td class="ie"></td>
                <td class="opera"></td>
                <td class="safari"></td>
            </tr>
        </tbody>
    </table>
</body>
</html>
```

在浏览器预览效果如图13-11所示。



图13-11　CSS Sprite实例

分析：

这个例子使用的“雪碧图”如图13-12所示。



图13-12　CSS Sprite实例用到的“雪碧图”

CSS Sprite技术最大的优点就是减少HTTP请求数，从而提高页面的加载速度。除了减少HTTP请求数，CSS Sprite还减小了图片整体的大小。一般情况下，几张小图片合并成一张大图片后的大小总是小于这几种小图标的大小总和。

不过CSS Sprite也存在很明显的缺点，那就是“开发和维护比较困难”。在开发过程中，需要精准计算每个小背景图在大背景图中的位置，比较繁琐复杂。在维护过程中，有时候需要增加新的小背景图，可能需要调整已有的小背景图位置，这样又得重新精准调整小背景图的位置。

在使用CSS Sprite技术时，我们需要注意以下几点。

（1）在开发后期而不是开发前期使用CSS Sprite。

“雪碧图”的制作比较繁琐，如果在开发前期就使用，可能雪碧图中小背景图位置得经常改变，维护起来比较麻烦。因此建议在开发后期再使用CSS Sprite。

（2）有条理地组织“雪碧图”。

我们将小背景图合并成“雪碧图”的时候，应该将小背景图按照类别、风格、大小等分门别类地放好，不要东放一个西放一个。有条理地组织“雪碧图”便于后期的维护。

（3）控制“雪碧图”的大小。

据研究表明，如果图片的大小在200KB以内，则图片传输的时间是差不多的。因此，“雪碧图”最好不要超过200KB。如果图片太大，会耗费更多的传输时间，从而影响网站加载速度。对于大于200KB的“雪碧图”，我们应该根据类别分割成多个“雪碧图”比较好。

13.4　Icon Font图标

对于小图标效果，估计很多人能够想到的是使用CSS Sprite结合背景图片来实现。但是在实际开发中，我们都是秉着“少用图片”的原则。因为图片数据传输量大，并且会引发一次HTTP请求。而且对于这些小图标，也可能会有多个尺寸、多种颜色等，如果要使用背景图片来实现非常麻烦。图13-3和图13-4分别为绿叶学习网中顶部效果和文章页面中的小图标。

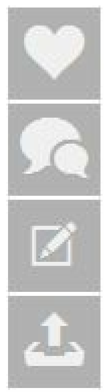


图13-13　绿叶学习网

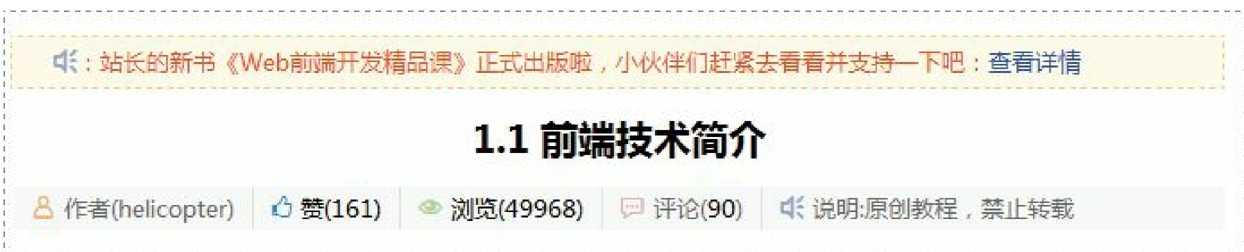


图13-14　绿叶学习网回顶部效果中的小图标 文章页面中的小图标

想要实现小图标效果，比较好的解决方法就是使用icon font图标技术。所谓的“icon font图标技术”，指的就是使用字体文件取代图片文件，来实现小图标效果。

13.4.1　iconfont网站

iconfont是国内功能很强大且图标内容很丰富的矢量图标库。它是

由阿里巴巴体验团队倾力打造的一款集设计和前端开发的便捷工具。iconfont提供矢量图标下载、在线存储、格式转换等功能。要想使用iconfont图标，就必须结合这个网站。官方网址为http://www.iconfont.cn/。

接下来，我们来简单介绍一下如何使用iconfont网站。

01

打开官网首页，界面如图13-15所示。左上角“图标库”下有两个选项，分别为“官方公开库”和“所有公开库”。



图13-15　第1步

02

单击“所有公开库”之后，界面如图13-16所示。右上角有四个排序选项，可以自行选择。

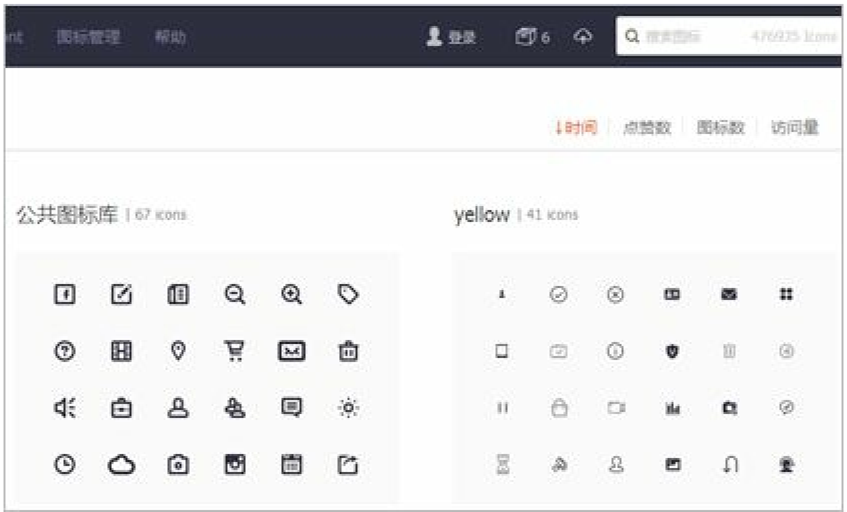


图13-16　第2步

03

任意单击一个图标库进去，界面如图13-17所示。



图13-17　第3步

04

单击你需要的图标，然后被单击的图标会收藏在右上角的“暂存架”上，如图13-18所示。



图13-18　第4步

05

选择完毕之后，在“暂存架”上单击“下载到本地”。解压文件夹之后，如图13-19所示。

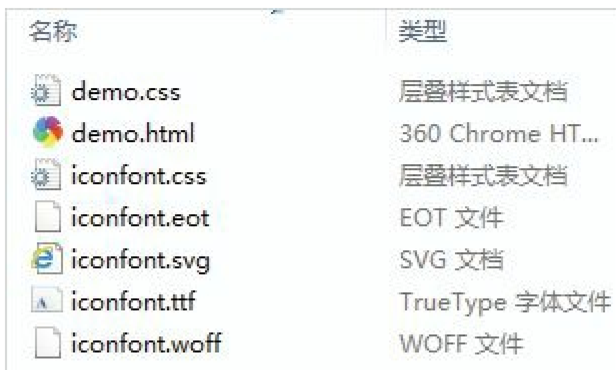


图13-19　第5步

13.4.2　icon font技术

从iconfont网站下载的文件中，有几个是比较重要的：iconfont.eot、iconfont.svg、iconfont.ttf、iconfont.woff，如表13-1所示。其实这四个都是字体文件，我们简单了解一下即可。

表**13-1** 字体文件格式

文件格式	说　明
.eot	微软开发的用于嵌入网页的字体，IE专用字体格式

.woff	W3C组织推荐的标准，Web字体最佳格式
.ttf	Mac OS和Windows操作系统中最常见的字体格式
.svg	W3C组织制定的开放标准的图形格式

想要使用iconfont图标，我们就必须要在CSS中引入这四个字体文件。

语法：

```
@font-face
{
    font-family: "iconfont";
    src: url("fonts/iconfont.eot"); /* IE9 兼容模式 */
    src: url("fonts/iconfont.eot?#iefix") format("embedded-opentype"),
                url("fonts/iconfont.woff") format("woff"),
                url("fonts/iconfont.ttf") format("truetype"),
                url("fonts/iconfont.svg") format("svg");
    font-weight: normal;
    font-style: normal;
}
.iconfont
{
    font-family: "iconfont";
    font-style: normal;
    font-weight: normal;
    -webkit-font-smoothing:antialiased;
    -moz-osx-font-smoothing:grayscale;
    }
```

说明：

使用iconfont图标的关键语法就是@font-face。@font-face用于自定义字体，具体语法不在此展开，可以参考绿叶学习官网的CSS3教程中“嵌入字体@font-face”这一节。“-webkit-font-smoothing:antialiased;”用于webkit引擎浏览器中的抗锯齿，而“-moz-osx-font-smoothing:grayscale;”用于Mac OSX Firefox浏览器中的抗锯齿。

上面代码看似很复杂，但是我们也没必要记住。在实际开发中，我们直接复制过去用就行了。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        /*自定义图标字体*/
        @font-face
        {
            font-family: "iconfont";
            src: url("fonts/iconfont.eot"); /* IE9 兼容模式 */
            src: url("fonts/iconfont.eot?#iefix") format("embedded-opentype"
                         url("fonts/iconfont.woff") format("woff"),
                         url("fonts/iconfont.ttf") format("truetype"),
                         url("fonts/iconfont.svg") format("svg");
            font-weight: normal;
            font-style: normal;
        }
        .iconfont
        {
              font-family: "iconfont";
              font-style: normal;
              font-weight: normal;
              -webkit-font-smoothing:antialiased;
              -moz-osx-font-smoothing:grayscale;
        }
    </style>
</head>
<body>
    <i class="iconfont">&#xe654;</i>
</body>
```

```
</html>
```

在浏览器预览效果如图13-20所示。

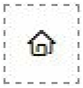


图13-20　iconfont小图标实例（1）

分析：

小伙伴们可能会觉得很奇怪，“”这个是什么？其实从iconfont网站下载下来的文件中有一个“demo.html”，我们点击进去可以看到每个图标都有对应一个字符串。我们想要使用哪个小图标，就写上对应的字符串即可，如图13-21所示。



图13-21　图标对应的字符串

也就是说，如果我们想要在页面中使用iconfont图标，必须要有四步。

（1）下载好图标字体文件并且放入网站目录中。

（2）在CSS中，使用@font-face自定义字体。

（3）在HTML中，元素添加class="iconfont"。

（4）在元素中添加图标对应的字符串。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title></title>
    <style type="text/css">
          @font-face
          {
              font-family: "iconfont";
              src: url("fonts/iconfont.eot"); /* IE9 兼容模式 */
              src: url("fonts/iconfont.eot?#iefix") format("embedded-opentyp
                          url("fonts/iconfont.woff") format("woff"),
                          url("fonts/iconfont.ttf") format("truetype"),
                          url("fonts/iconfont.svg") format("svg");
              font-weight: normal;
              font-style: normal;
        }
        .iconfont
        {
            font-family: "iconfont";
            font-style: normal;
            font-weight: normal;
            -webkit-font-smoothing:antialiased;
            -moz-osx-font-smoothing:grayscale;
        }
        i
        {
            font-size:60px;
            font-weight:bold;
            color:Red;
        }
    </style>
</head>
<body>
    <i class="iconfont">&#xe654;</i>
</body>
</html>
```

在浏览器预览效果如图13-22所示。



图13-22　iconfont小图标实例（2）

分析：

我们还可以在CSS中为iconfont图标定义font-size、font-weight、color等，是不是感觉控制小图标的外观跟控制字体的外观一样的简单方便？

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
          @font-face
          {
              font-family: "iconfont";
              src: url("fonts/iconfont.eot"); /* IE9 兼容模式 */
              src: url("fonts/iconfont.eot?#iefix") format("embedded-opentyp
                           url("fonts/iconfont.woff") format("woff"),
                           url("fonts/iconfont.ttf") format("truetype"),
                           url("fonts/iconfont.svg") format("svg");
              font-weight: normal;
              font-style: normal;
        }
        .iconfont
        {
              font-family: "iconfont";
              font-style: normal;
```

```
            font-weight: normal;
            -webkit-font-smoothing:antialiased;
            -moz-osx-font-smoothing:grayscale;
        }
        ul
        {
            font-size:16px;
            font-weight:bold;
        }
        li{color:Blue;}
        i{font-size:21px;color:Red;margin-right:5px;}
    </style>
</head>
<body>
    <ul>
        <li><i class="iconfont">&#xe654;</i>回到首页</li>
        <li><i class="iconfont">&#xe655;</i>搜索一下</li>
        <li><i class="iconfont">&#xe656;</i>个人中心</li>
        <li><i class="iconfont">&#xe653;</i>我的购物</li>
        <li><i class="iconfont">&#xe65d;</i>我的菜单</li>
    </ul>
</body>
</html>
```

在浏览器预览效果如图13-23所示。



图13-23　Icon Font小图标实例（3）

对于iconfont图标技术，还有两点补充说明。

（1）iconfont官网除了提供字体图标，还提供图标管理、WebFont技术等服务。我们应该到官网自己摸索一下，了解这些会给你的前端开发带来很大的帮助；

（2）对于字体图标，如果iconfont官网满足不了你的需求，我们可以访问国外最大的图标分享网站icomoon。相对于iconfont，icomoon的

图标种类更多，功能也更为强大。毕竟这是前端界最好的一个图标分享网。icomoon官网地址为https://icomoon.io/。

第14章　重要概念

14.1　CSS中的重要概念

在最后一章里，我们给大家介绍一下CSS中几个极其重要的概念。了解这些概念对你深入理解CSS的本质相当重要：

（1）包含块containing block。

（2）BFC和IFC。

（3）层叠上下文。

这些东西比较抽象，难以理解。虽然国内甚少书籍会涉及，但是这些概念在CSS中却扮演着非常重要的角色，不要觉得难就跳过。如果要真正地掌握CSS，大家一定认真研读、一定要认真研读、一定要认真研读（重要的事情说三遍）。

认真理解这些，可以帮助你彻底地精通CSS。

14.2　包含块

14.2.1　什么是包含块

我们都知道，如果有两个div，一个是父元素，另外一个是子元素，父元素会决定子元素的大小和定位。包含块是什么呢？简单来说，就是可以决定一个元素大小和定位的元素。

包含块是视觉格式化模型中的一个重要概念，它与CSS盒子模型类似。包含块也可以理解为一个矩形盒子，这个矩形的作用是为这个矩形内部的后代元素（子元素、孙元素等）提供一个参考。一个元素的大小和定位往往是由该元素所在的包含块决定的。

通常情况下，一个元素的包含块是由离它最近的“块级祖先元素”的“内容边界”决定的。但当元素被设置为绝对定位时，此时该元素的包含块是由离它最近的“position:relative”或“position:absolute”的祖先元素决定。一个元素生成的盒子会扮演该元素的内部元素包含块的角色。也就是说，一个元素的CSS盒子为它的内部元素建造了包含块。

14.2.2　包含块的判定以及包含块的范围

一个元素会为它的内部元素创建包含块，内部元素的大小以及定位都跟它的包含块有关。那么是不是说一个元素的包含块就是它的父元素呢？答案是否定的。

1．根元素

根元素（HTML元素），是一个页面中最顶端的元素，它没有父元素。根元素存在的包含块，被称为“初始包含块（initial containing block）。

2．固定定位元素

如果元素的position属性为fixed，那么它的包含块是当前可视窗

口，也就是当前浏览器窗口。

3．静态定位元素和相对定位元素

如果元素的position属性为static或relative，那么它的包含块是它最近的块级祖先元素创建的。祖先元素必须是block、inline-block或者table-cell类型。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
</head>
<body>
    <div>
        <p><span></strong>绿叶学习网</span><strong></p>
    </div>
</body>
</html>
```

对于上面的HTML元素的包含块关系如表14-1所示。这里注意一下，根据上面的定义，strong的包含块是p，而不是span。

表**14-1** 包含块关系表

元　　素	包　含　块
div	body
p	div
span	p

strong	p

4. 绝对定位元素

如果元素的position属性为absolute，那么它的包含块是由最近的position属性不为static的祖先元素。这里的祖先元素可以是块元素，也可以是行内元素。

从上面我们知道，绝对定位元素是根据其包含块来定位的，这个包含块是离它最近的position属性不为static的祖先元素。如果绝对定位元素找不到position属性不为static的祖先元素，则它的包含块是根元素（html元素）。现在我们知道为什么在默认情况下，绝对定位元素是相对浏览器窗口来定位的了吧？

对于包含块的范围，我们也分两种情况考虑。

（1）如果祖先元素是块元素，则包含块的范围为祖先元素的padding edge形成。

（2）如果祖先是行内元素，则包含块取决于祖先元素的direction属性。

当祖先为行内元素时，包含块的范围判定比较复杂，而且也没多少用处，此处不详细展开，可以参考：http://www.w3help.org/zh-cn/kb/008/。

在“深入定位”这一节提到的“绝对定位元素是相对于外层第一个设置了‘position:relative;’‘position:absolute;’或‘position:fixed’的祖先元素来进行定位的。”其实就是跟绝对定位元素的包含块有关。

14.3 层叠上下文

层叠上下文也许我们比较少接触，但这却是一个非常重要的概念。理解层叠上下文，不仅可以帮助我们深入理解z-index对元素堆叠顺序的控制，而且对于我们深入理解浮动和定位也是非常重要的。这一节我们只针对CSS2.1而言，对于CSS3新环境下层叠上下文的变化，我们不在此展开介绍。

在这一节，我们需要认真理解两个概念。

（1）层叠上下文（stacking context）。

（2）层叠级别（stacking level）。

14.3.1 什么是层叠上下文?

层叠上下文，是HTML中的一个三维的概念。从“z-index属性”这一节我们知道，虽然一个网页是平面的，但实际上网页是三维结构，除了x轴、y轴，它还有z轴。z轴往往都是用来设定层的先后顺序的。

层叠上下文跟块级格式上下文（BFC）相似，是可以创建出来的。也就是说，跟创建BFC一样，你可以在CSS中添加一定的属性来将某个元素创建一个层叠上下文出来。

如果一个元素具备以下任何一个条件（不考虑CSS3），则该元素会创建一个新的层叠上下文。

（1）根元素。

（2）z-index不为auto的定位元素。

这里注意一下，根元素会创建一个层叠上下文，我们称之为“根层叠上下文”。这个与根元素创建一个BFC是一样的。对于根层叠上下文，没什么好讲的。

从上面我们知道，如果我们想要创建一个新的层叠上下文，也就只

有一个途径了——z-index属性。

14.3.2　什么是层叠级别？

层叠级别，即stacking level。从上面我们知道，可以使用z-index属性为一个元素创建一个新的层叠上下文。但一个元素往往会有背景色、浮动子元素、定位子元素等，那么这些东西又是遵循着怎样的顺序来堆叠的呢？

同一个层叠上下文的背景色以及内部元素，谁在上谁在下，这些都是由层叠级别”来决定的。也就是说，层叠级别是针对同一个层叠上下文而言的。层叠级别与层叠上下文，是两个不同的概念，大家要认真理解好。

在同一个层叠上下文中，层叠级别从低到高排列，如图14-1所示。

（1）背景和边框（父级）：也就是当前层叠上下文的背景和边框。

（2）负z-index：z-index为负值的“内部元素”。

（3）块盒子：普通文档流下的块盒子（block-level box）。

（4）浮动盒子：非定位的浮动元素（也就是排除了position:relative的浮动盒子）。

（5）行内盒子：普通文档流下的行内盒子（inline-level box）。

（5）z-index:0：z-index为0的“内部元素”。

（6）正z-index：z-index为正值的“内部元素”。

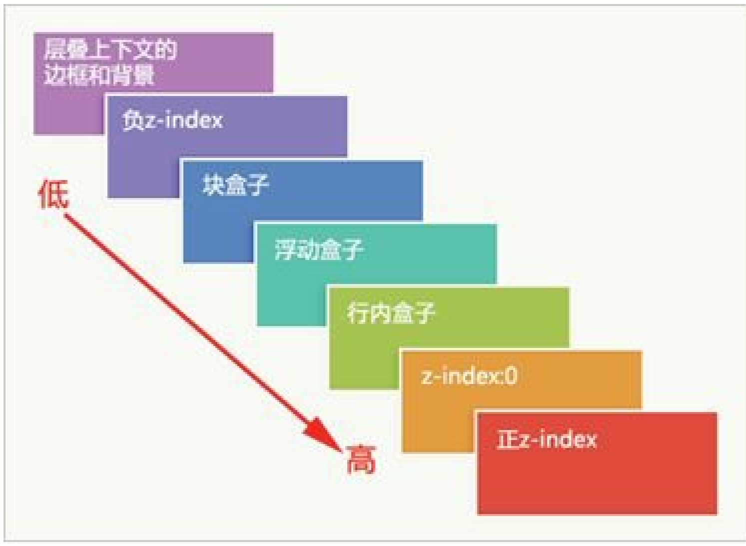


图14-1　同一层叠上下文中的“层叠级别”

从上图我们可以得到以下结论。

（1）除了“背景和边框”这一条是当前层叠上下文之外，其他的都是针对当前层叠上下文内部的元素。

（2）关于块盒子（block-level box）和行内盒子（inline-level box），我们在下一节“BFC和IFC”中会给大家介绍。注意，inline-block元素不是块盒子，而是行内盒子。

（3）父元素内部的元素（即后代元素），如果它是一个z-index取值不为auto的定位元素，则这个元素会创建新的层叠上下文。不过这个由内部元素创建的层叠上下文依旧属于父层叠上下文的一部分。也就是说，层叠上下文是可以嵌套的，内部元素所创建的层叠上下文均受制于父元素创建的层叠上下文。

这里问大家一个问题，为什么行内元素的层叠级别要比浮动元素和块元素的高呢？我明明感觉浮动元素和块元素要更高一点啊。我们先来看一个分析图如图14-2所示。

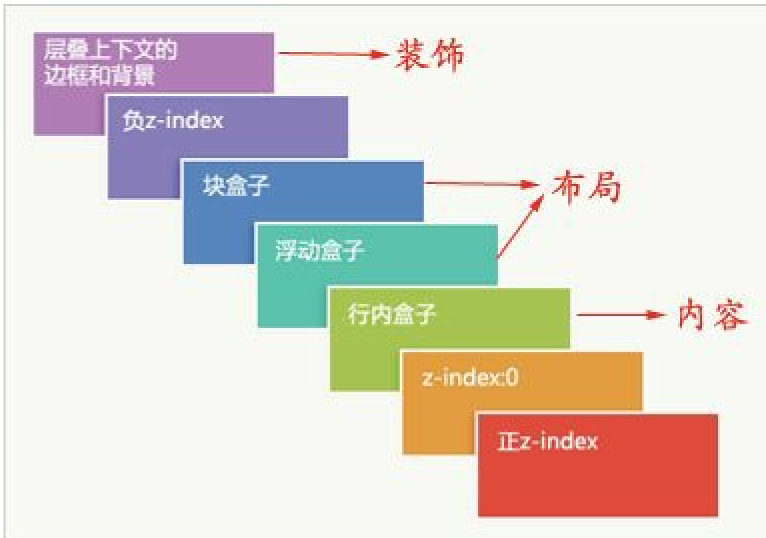


图14-2　层叠级别分析图

背景和边框一般为装饰属性，所以层叠级别最低。浮动元素和块元素一般用作布局，而行内元素都是内容。对于一个页面来说，最重要的当然是内容。因此，一定要让内容的层叠级别相当高。

14.3.3　层叠上下文的特点

一个元素在z轴方向上的堆叠顺序，是由“层叠上下文”和“层叠级别”这两个因素决定的：

（1）同一个层叠上下文中，我们比较的是“内部元素层叠级别”。层叠级别大的元素显示在上，层叠级别小的元素显示在下。

（2）同一个层叠上下文中，如果两个元素的层叠级别相同（即z-index值相同），则后面的元素堆叠在前面元素的上面，遵循“后来者居上”原则。

（3）不同的层叠上下文中，我们比较的是“父级元素层叠级别”。元素显示顺序以“父级层叠上下文”的层叠级别来决定显示的先后顺序，与自身的层叠级别无关。

举例：

分析：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        #wrapper
        {
            width:400px;
            height:200px;
            border:1px solid gray;
            padding:10px;
        }
        img{float:left;}
        #content{background-color:#FFACAC;}
    </style>
</head>
<body>
    <div id="wrapper">
        <img src="images/ailianshuo.png" alt=""/>
        <div id="content">水陆草木之花，可爱者甚蕃。晋陶渊明独爱菊。自李唐来，世
    </div>
</body>
</html>
```

在浏览器预览效果如图14-3所示。

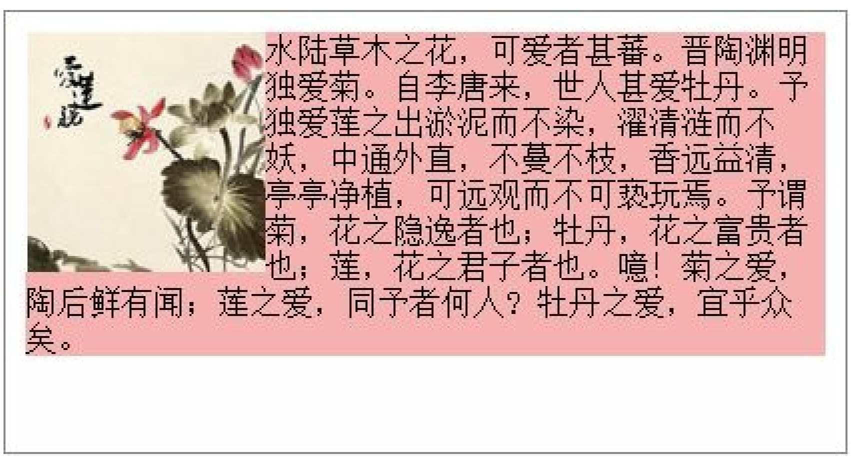


图14-3　浮动引起的文字环绕效果

分析：

一个元素浮动之后，它的层叠级别比普通文档流的块级盒子的层叠级别要高。此时浮动元素会“浮”到上面去，脱离文档流。三维立体分析图如图14-4所示。



图14-4　浮动引起的文字环绕效果分析图

这就是浮动的效果，大家一定要回顾“浮动布局”这一章中的例子，结合层叠、深入理解，便能了解其中奥秘。

14.4 BFC和IFC

14.4.1 基本概念

在CSS中，页面中任何一个元素都可以看成是一个盒子。在普通文档流（normal flow）中，盒子会参与一种格式上下文（formatting context）。这个盒子可能是块盒子（block-level box），也可能是行内盒子（inline-level box）。一个盒子只能是块盒子或者是行内盒子，不能同时是块盒子又是行内盒子。其中块盒子参与BFC（块级格式上下文），行内盒子参与IFC（行级格式上下文）。

1．格式上下文（formatting context）

格式上下文是W3C CSS2.1 规范中的一个重要概念。它指的是页面中的一块渲染区域，并且这个格式上下文有一套自己的渲染规则。格式上下文决定了其内部元素将如何定位，以及和其他元素之间的关系。

格式上下文有两种。

（1）块级格式上下文（Block Formatting Context，BFC）。

（2）行级格式上下文（Inline Formatting Context，IFC）。

2．盒子（box）

盒子，又称CSS盒子，是CSS布局的基本单位。简单来说，一个页面就是由很多个盒子组成的（具体请参考CSS盒子模型）。元素的类型和display属性决定了盒子的类型。不同类型的盒子，会参与不同的格式上下文。表14-2列出了常见的display属性。

表**14-2** 常见的**display**属性取值

属　性　值	说　　明

inline	行内元素
block	块元素
inline-block	行内块元素
table	以表格形式显示，类似于table元素
table-row	以表格行形式显示，类似于tr元素
table-cell	以表格单元格形式显示，类似于td元素
none	隐藏元素

上面已经提到了，盒子的类型一般有“块盒子”和“行内盒子”这两种。

（1）块盒子。

块盒子，即block-level box。元素类型（即display属性）为block、table、list-item的元素，会生成块盒子。

块盒子会参与块级格式上下文。也就是说，元素类型为block、table、list-item的元素都会参与块级格式上下文（BFC）。

（2）行内盒子。

行内盒子，即inline-level box。元素类型（即display属性）为inline、inline-block、inline-table的元素，会生成行内盒子。

行内盒子会参与行级格式上下文。也就是说，元素类型为inline、inline-block、inline-table的元素都会参与行级格式上下文（IFC）。

除了block-level box和inline-level box这两种盒子，在CSS中还有一

个run-in box盒子。不过run-in box是CSS3新增的，在此不介绍。

14.4.2　什么是BFC

BFC，全称Block Formatting Context（块级格式上下文）。它是一个独立的渲染区域，只有块盒子（block-level box）参与。块级格式上下文规定了内部的块盒子是如何布局的，并且这个渲染区域与外部区域毫不相关。

1．如何创建BFC

W3C标准中对BFC的定义：浮动元素，绝对定位元素（position为absolute或fixed），元素类型（即display属性）为inline-block、table-caption、table-cell，以及overflow属性不为visible的元素将会创建一个新的块级格式上下文（BFC）。

如果一个元素具备以下任何一个条件，则该元素都会创建一个新的BFC。

（1）根元素。

（2）float属性除了none以外的值，也就是“float:left”和“float:right”。

（3）position属性除了static和relative以外的值，也就是“position:absolute”和“position:fixed”。

（4）overflow属性除了visible以外的值，也就是“overflow:auto”“overflow:hidden”和“overflow:scroll”。

（5）元素类型（即display属性）为inline-block、table-caption、table-cell。

也就是说，如果我们要创建一个新的BFC，只需要在CSS中添加上面任意一个属性即可。创建新的BFC可以帮助我们解决很多问题，下面我们会详细介绍。这里要注意一下，根元素也会创建BFC，在默认情况下一个页面中所有的元素都处于同一个BFC中。这是默认的，不需要我们去设置。不过理解这一点对于我们后面理解很多东西十分重要。虽然

这些属性都可以创建BFC，但是也会产生一些效果。

（1）float:left和float:right会将元素移到左边或右边，并被其他元素环绕。

（2）overflow:hidden会将超出元素的内容隐藏。

（3）overflow:scroll会产生多余的滚动条。

（4）display:table可能引发响应性问题。

……

因此如果我们要创建一个BFC，一定要根据需求来选择最恰当的属性。

这里要注意一下，类型为flex和inline-flex的元素也会创建BFC，只不过这些是CSS3的内容，我们在此忽略。此外根据定义，类型为block、table的元素不会创建BFC。小伙伴们可能就有疑问了，为什么block类型元素不会创建BFC啊？最开始提到：元素类型（即display属性）为block、table、list-item的元素，会生成块盒子（block-level box），然后块盒子会参与BFC。其实从这句话我们已经得到明确答案了：block、table、list-item等类型的元素的是参与BFC，而不是创建BFC。

2．BFC的特点

W3C标准描述BFC的特点共有两条：

（1）在一个BFC中，盒子从顶端开始垂直一个接着一个地排列，两个相邻盒子之间的垂直间距由margin属性决定。在同一个BFC中，两个相邻块盒子之间垂直方向上的外边距会叠加。

（2）在一个BFC中，每一个盒子的左外边界（margin-left）会紧贴着容器的左边（border-left）（对于从右到左的格式化，则相反），即使存在浮动元素也是如此。

从上面的W3C标准定义，我们可以得出以下几点重要结论（非常重

要，请字斟句酌地理解记忆）。

（1）在一个BFC内部，盒子会在垂直方向上一个接着一个地排列。

（2）在一个BFC内部，相邻的margin-top和margin-bottom会叠加。

（3）在一个BFC内部，每一个元素的左外边界会紧贴着包含盒子的左边，即使存在浮动也是如此。

（4）在一个BFC内部，如果存在内部元素是一个新的BFC，并且存在内部元素是浮动元素。则该BFC的区域不会与float元素的区域重叠。

（5）BFC就是页面上的一个隔离的盒子，该盒子内部的子元素不会影响到外面的元素。

（6）计算一个BFC的高度时，其内部浮动元素的高度也会参与计算。

有些新手觉得很奇怪，为什么在一个BFC中，盒子会在垂直方向上一个接着一个排列呢？如果在一个BFC中有一个盒子是span这种行内元素，岂不是不符合了？再次提醒一下，能够参与BFC中的盒子是块盒子（block-level box）。就算在这个BFC中存在一个行内元素，这个行内元素参与的是IFC而不是BFC，别搞混了。

14.4.3　BFC的用途

上面给大家介绍了BFC的特点以及如何创建一个新的BFC。说了那么多，那BFC究竟有什么用呢？BFC的用途很多，常见的有三个。

（1）创建BFC来避免垂直外边距叠加。

（2）创建BFC来清除浮动。

（3）创建BFC来实现自适应布局。

1．创建BFC来避免垂直外边距叠加

在之前的章节里已经给大家详细介绍过了外边距叠加的问题。外边距叠加，准确地说，是在同一个BFC中，两个相邻的margin-top和margin-bottom相遇时，这两个外边距将会合并为一个外边距，即“二变一”。其中，叠加之后的外边距高度等于发生叠加之前的两个外边距中最大值。之所以垂直外边距会发生叠加，其实这都是BFC的特点。学到这里，相信大家都会有“柳暗花明又一村”的感觉。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        #wrapper
        {
            width:200px;
            border:1px solid gray;
            overflow:hidden;/*创建一个新的BFC*/
        }
        #a,#b
        {
            height:60px;
            line-height:60px;
            text-align:center;
            font-size:30px;
            color:White;
            background-color:Purple;
        }
        #a{margin-bottom:20px;}
        #b{margin-top:30px;}
    </style>
</head>
<body>
    <div id="wrapper">
        <div id="a">A</div>
        <div id="b">B</div>
    </div>
</body>
</html>
```

在浏览器预览效果如图14-5所示。

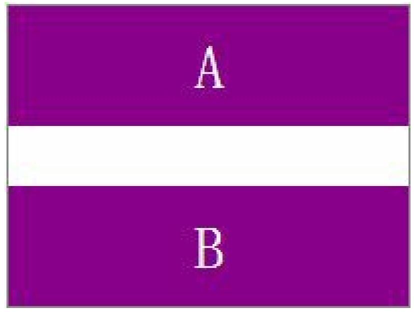


图14-5　垂直外边距叠加

分析：

在这个例子中，我们使用“overflow:hidden”为父元素创建一个BFC，也就是说父元素是一个BFC了。因此A和B位于同一个BFC中。

A的margin-bottom为20px，B的margin-top为30px。由于在同一个BFC中，相邻两个盒子的垂直外边距会叠加，因此A和B的垂直距离为30px。

细心的小伙伴们就有疑问了：我不给父元素添加“overflow:hidden”来创建新的BFC，垂直外边距也会发生叠加，这是什么情况？大家别忘了根元素本身就是一个BFC，如果我们没有为父元素创建BFC，则默认情况下A和B就是处于根元素的BFC中。

在实际开发中，如果我们想要避免垂直外边距叠加，怎么办呢？根据第二个结论“同一个BFC中，相邻的margin-top和margin-bottom会叠加”，既然相邻的margin-top和margin-bottom必须处于同一个BFC才会发生叠加，那么我们把这两个元素放在不同的BFC中，就可以解决了。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        #wrapper
        {
            width:200px;
            border:1px solid gray;
            overflow:hidden;/*创建一个BFC*/
        }
        #bfc-box
        {
            overflow:hidden;/*创建一个BFC，避免外边距叠加*/
        }
        #a,#b
        {
            height:60px;
            line-height:60px;
            text-align:center;
            font-size:30px;
            color:White;
            background-color:Purple;
        }
        #a{margin-bottom:20px;}
        #b{margin-top:30px;}
    </style>
</head>
<body>
    <div id="wrapper">
        <div id="a">A</div>
        <div id="bfc-box">
            <div id="b">B</div>
        </div>
    </div>
</body>
</html>
```

在浏览器预览效果如如图14-6所示。

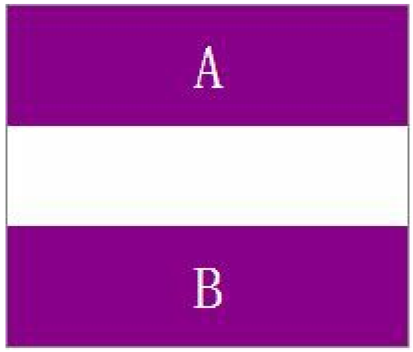


图14-6　创建BFC避免垂直外边距叠加

分析：

在这个例子中，A和B处于不同的BFC，其中A处于#wrapper元素的BFC中，B处于#bfc-box元素的BFC中，所以不会发生垂直外边距叠加。这里注意一下，假如我们不给#bfc-box元素添加“overflow:hidden”，A和B也会发生垂直外边距叠加。此时A和B都不属于相邻的元素，它们为什么还会发生外边距叠加呢？

我们再来看看第二个结论“同一个BFC中，相邻的margin-top和margin-bottom会叠加”，这里的相邻不是指“相邻的兄弟元素”，而是指相邻的margin-top和margin-bottom。

2．创建BFC来清除浮动

（1）BFC包含浮动。

我们都知道可以使用“overflow:hidden”来清除浮动，但很少知道为什么。根据第六个结论“计算BFC的高度时，其内部浮动子元素的高度也会参与计算”可以知道，如果一个元素是一个BFC，则计算该元素高度的时候，内部浮动子元素的高度也得算进去。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        #wrapper
        {
            width:200px;
            border: 1px solid black;
        }
        #first,#second
        {
            width:80px;
            height:40px;
            border:1px solid red;
        }
        #first{float:left;}
        #second{float:right;}
    </style>
</head>
<body>
    <div id="wrapper">
        <div id="first"></div>
        <div id="second"></div>
    </div>
</body>
</html>
```

在浏览器预览效果如图4-7所示。

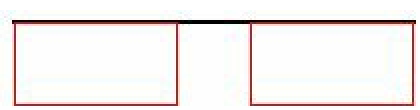


图14-7　浮动引起的父元素高度塌陷

分析：

在这个例子中，由于父元素没有定义高度，因此父元素无法把浮动子元素包裹起来，这就会造成父元素高度塌陷。如果我们给父元素添加“overflow:hidden”，在浏览器预览效果如图14-8所示。

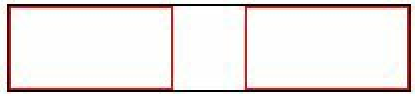


图14-8　创建BFC来清除浮动

这是因为“overflow:hidden”使得父元素变成了一个BFC，由于BFC在计算它自身高度的时候，会把浮动子元素的高度算进去，因此最终父元素的高度等于浮动子元素的高度。此时就相当于清除了浮动。当然我们也可以通过给父元素添加“display:inline-block”“float:left”等来创建新的BFC来实现浮动的清除。

创建BFC的方式很多，上面已经详细给大家介绍了。不过也明确说过，不同的属性会有不同的副作用。像这个例子，如果使用“overflow:scroll”确实是可以清除浮动，但是却无缘无故地增加了滚动条，这就不是我们想要的效果了（大家可以自行试试）。因此如果我们要创建一个BFC，一定要根据需求来选择最恰当的属性。

（2）BFC避免文字环绕。

有时浮动div旁边的文本会环绕它（如图14-9所示），但是有时候我们想要如图14-10的效果，我们可以通过创建BFC来实现。

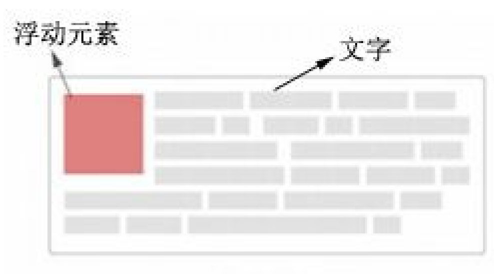


图14-9　文字环绕效果

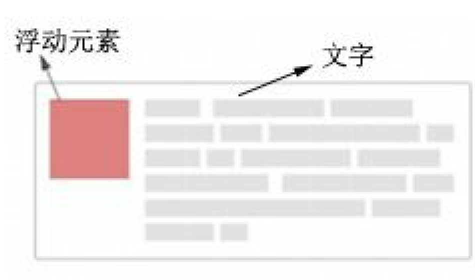


图14-10　预期效果

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        #wrapper
        {
            width:400px;
            height:200px;
            border:1px solid gray;
            padding:10px;
        }
        img{float:left;}
        #content{background-color:#FFACAC;}
    </style>
</head>
<body>
    <div id="wrapper">
        <img src="images/ailianshuo.png" alt=""/>
        <div id="content">水陆草木之花，可爱者甚蕃。晋陶渊明独爱菊。自李唐来，世
    </div>
</body>
</html>
```

在浏览器预览效果如图14-11所示。

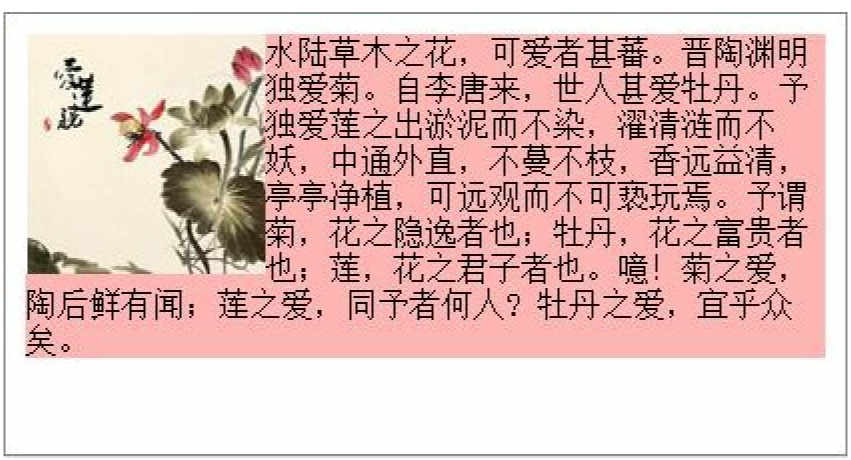


图14-11　浮动引起的文字环绕效果

分析：

根据层叠上下文的知识我们知道：一个元素浮动之后，它的层叠级别（stacking level）比普通文档流的块级盒子的层叠级别要高。此时浮动元素会“浮”到上面去，脱离文档流，如图14-12所示。



图14-12　浮动引起的文字环绕效果分析图

在这个例子中，我们为#content元素添加“overflow:hidden”，此时#content元素变成了一个新的BFC，在浏览器预览效果如图14-13所示。

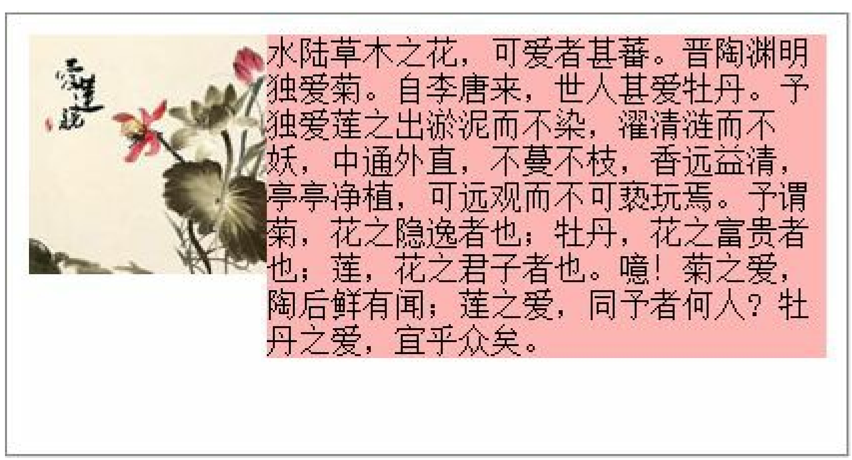


图14-13　创建BFC避免文字环绕

根据第四点结论：在一个BFC内部，如果存在内部元素是一个新的BFC，并且存在内部元素是浮动元素，则该BFC的区域不会与float元素的区域重叠。

3. 使用**BFC**创建自适应两列布局

自适应两列布局，指的是在左右两列中，其中有一列的宽度为自适应，另外一列宽度是固定的。在之前的章节中，我们介绍过使用负margin技术来实现自适应左右两列布局。这里我们介绍另外一种实现方式，那就是使用BFC创建自适应两列布局。

举例：

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
    <style type="text/css">
        #sidebar
        {
            float:left;
            width:100px;
            height:150px;
            background:#FF6666;
        }
        #content
        {
            height:200px;
            background-color:#FFCCCC;
        }
    </style>
</head>
<body>
    <div id="sidebar"></div>
    <div id="content"></div>
</body>
</html>
```

在浏览器预览效果如图14-14所示。

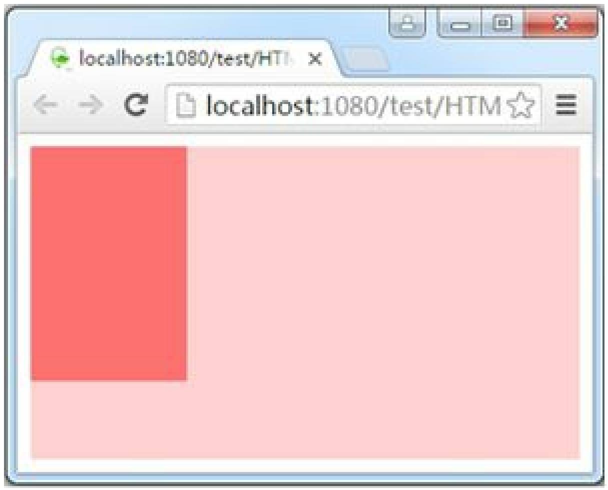


图14-14　浮动引起的元素覆盖效果

分析：

根据层叠上下文的知识我们知道：一个元素浮动之后，它的层叠级别（stacking level）比普通文档流的块级盒子的层叠级别要高。此时浮动元素会“浮”到上面去，脱离文档流，如图14-15所示。



图14-15　浮动引起的元素覆盖效果分析图

上面的预览结果，刚好满足了结论的第三点“在一个BFC中，内部每一个元素的左外边界会紧贴着包含盒子的左边，即使存在浮动也是如此。”

在这个例子中，我们为#content元素添加“overflow:hidden”，此时

#content元素变成了一个新的BFC，在浏览器预览效果如图14-16所示。

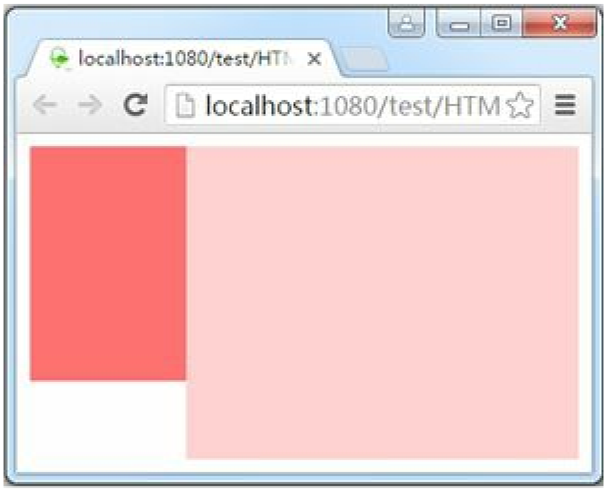


图14-16　创建BFC创建自适应两列布局

我们改变浏览器的宽度，就很容易可以看出自适应左右两列布局实际效果是怎样的。这个例子跟“BFC避免文字环绕”的例子是一样的道理。

这一节的内容非常复杂，很多东西大家一下子理解不来。不过没关系，回头多看几次就行。此外，对于BFC和IFC这些概念的学习，我们应该去寻找官方的定义，因为这是最权威和最准确的。

后记

看到这里，相信小伙伴们在HTML和CSS方面已经走了很远。本书是我多年开发的心血总结，基本毫无保留地分享出来。这也是希望小伙伴们少走弯路，在前端之路走得更快更远。人生苦短，时间更多应该用来追逐自己喜欢的东西，而不是在一些弯路上白白浪费。

从入门到进阶，大家已经系统学了一遍HTML和CSS的基础知识和开发技巧。接下来，就看我们在实际开发中怎么运用了。在进阶的学习中，当你为那些高级技巧所折服时，还是要认真说一句：HTML和CSS进阶注重的并不是学到了多少开发技巧，而是学会了代码规范以及性能优化等，毕竟网站的性能与可维护性是才是最重要的。

在HTML和CSS进阶的学习中，我们应该都清楚地知道：每一门Web技术都不简单，我们平常应该深入地去学习技巧和提高技能。别每次一出问题就赖浏览器有bug。事实上，浏览器倒不会有bug，而是你的心态有bug。

学完HTML和CSS进阶，建议大家下一步先去学习CSS3的内容。CSS3相当强大，可以实现非常棒的效果，一定要去看看。此外，平常我们也应该多查看大型互联网公司的源代码，就算你会做，也可以从中学到很多东西。我们常说注重实践，其实通过查看别人的源码来学习，也是实践的一部分。

当然，通过本书我们只是学习了HTML和CSS。然而Web前端开发技术远不止这些，如果小伙伴们想要成为一名合格的前端开发人员，我们接下来要学习更多前端技术，例如JavaScript、jQuery、HTML5、CSS3等。对于更多的前端技术，可以到绿叶学习网（http://www.lvyestudy.com）或发邮件（lvyestudy@foxmail.com）与我交流。

欢迎来到异步社区！

异步社区的来历

异步社区(www.epubit.com.cn)是人民邮电出版社旗下IT专业图书旗舰社区，于2015年8月上线运营。

异步社区依托于人民邮电出版社20余年的IT专业优质出版资源和编辑策划团队，打造传统出版与电子出版和自出版结合、纸质书与电子书结合、传统印刷与POD按需印刷结合的出版平台，提供最新技术资讯，为作者和读者打造交流互动的平台。



异步社区
技术圈
图书
电子书
文章
图书/作者/类别
写作
我们一步啦
异步社区成立一周年大型活动开启
我要写书
近期活动
+更多
异步社区成立一周年大型赠书活动开启！
异步社区的来历 异步社区是人民邮电出版社旗下IT 专，业图书旗舰社区，于2015 年8 月上线运营。异步社区依托于人民邮电出版社 20余 年的IT 专业...
猫叔郭志敏 2016-08-02
阅读 575 推荐 2 收藏 0 评论 8
2016 iWeb峰会北京站即将开启，为HTML5嘟喊！
每一次振臂高呼辐射行业的影响， 每一天无数人兢兢业业的勤奋，2016雄起！ 来吧，8月27日，HTML5峰会北京站，我在这里，等你来，为HTML5嘟喊！...
猫叔郭志敏 2016-07-29
阅读 60 推荐 1 收藏 0 评论 0
周年庆满减促销 | 满100元减20元、满150元减35元、满200元减50元
+更多
CCIE路由和交换认证考试指南（第5版）（第1卷）
数据科学实战手册（R+Python）
软技能：代码之外的生存指南
Python密码学编程
Python游戏编程快速上手
机器学习项目开发实战
树莓派Python编程入门与实战（第2版）
像计算机科学家一样思考Python（第2版）
每周半价电子书
+更多
树莓派Python编程入门与实战（第2版）
[美] Richard Blum 勃鲁姆, Christine Bresnahan 布莱斯纳罕 (作者) 陈晓明 马立新 (译者)

社区里都有什么？

购买图书

我们出版的图书涵盖主流IT技术，在编程语言、Web技术、数据科学等领域有众多经典畅销图书。社区现已上线图书1000余种，电子书400多种，部分新书实现纸书、电子书同步出版。我们还会定期发布新书书讯。

下载资源

社区内提供随书附赠的资源，如书中的案例或程序源代码。

另外，社区还提供了大量的免费电子书，只要注册成为社区用户就可以免费下载。

与作译者互动

很多图书的作译者已经入驻社区，您可以关注他们，咨询技术问题；可以阅读不断更新的技术文章，听作译者和编辑畅聊好书背后有趣的故事；还可以参与社区的作者访谈栏目，向您关注的作者提出采访题目。

灵活优惠的购书

您可以方便地下单购买纸质图书或电子图书，纸质图书直接从人民邮电出版社书库发货，电子书提供多种阅读格式。

对于重磅新书，社区提供预售和新书首发服务，用户可以第一时间买到心仪的新书。

用户帐户中的积分可以用于购书优惠。100积分=1元，购买图书时，在 里填入可使用的积分数值，即可扣减相应金额。

> 特别优惠
>
> 购买本电子书的读者专享异步社区优惠券。 使用方法：注册成为社区用户，在下单购书时输入“**57AWG**”，然后点击“使用优惠码”，即可享受电子书8折优惠（本优惠券只可使用一次）。

纸电图书组合购买

社区独家提供纸质图书和电子书组合购买方式，价格优惠，一次购买，多种阅读选择。

社区里还可以做什么？

提交勘误

您可以在图书页面下方提交勘误，每条勘误被确认后可以获得100积分。热心勘误的读者还有机会参与书稿的审校和翻译工作。

写作

社区提供基于Markdown的写作环境，喜欢写作的您可以在此一试身手，在社区里分享您的技术心得和读书体会，更可以体验自出版的乐趣，轻松实现出版的梦想。

如果成为社区认证作译者，还可以享受异步社区提供的作者专享特色服务。

会议活动早知道

您可以掌握IT圈的技术会议资讯，更有机会免费获赠大会门票。

加入异步

扫描任意二维码都能找到我们：

异步社区

微信订阅号

微信服务号

官方微博

QQ群：368449889

社区网址：www.epubit.com.cn

官方微信：异步社区

官方微博：@人邮异步社区，@人民邮电出版社-信息技术分社

投稿**&**咨询：contact@epubit.com.cn